Kapitel 1

Einleitung

Der Entwurf von Benutzungsoberflächen für interaktive Software-Systeme gewinnt im Rahmen der Systementwicklung immer größere Bedeutung. Erfahrungen aus einer Vielzahl von Projekten [Myers 92a] zeigen, daß häufig mehr als die Hälfte des gesamten Entwicklungsaufwands für die Gestaltung der Benutzungsoberfläche aufgebracht wird.

Um die Konstruktion von Benutzungsoberflächen zu unterstützen, wurden in den letzten Jahren eine Vielzahl von Entwicklungswerkzeugen und -umgebungen realisiert. Diese sollen es den Entwicklern gestatten, den Entwurf ihrer Oberflächen auf einer Abstraktionsebene zu machen, die von der Hardware-Umgebung und von der Betriebssystem-Software unabhängig ist.

Mit der gräßeren Verbreitung von grafikfähigen Arbeitsstationen und PCs hat sich aber auch noch ein anderer Wandel vollzogen. Während früher die Betonung darauf lag, die Mäglichkeiten eines Software-Systems dem Benutzer zugänglich zu machen, rücken heute ergonomische Fragestellungen immer stärker in den Vordergrund. Die ergonomische Gestaltung von Benutzungsoberflächen ist jedoch eine Aufgabe, die nicht ohne die Beteiligung der späteren Benutzer erfolgreich bewältigt werden kann. Erschwert wird diese Aufgabe dadurch, daß auch die Anwender meist nicht in der Lage sind, ihre Gestaltungvorstellungen zu artikulieren und abzuschätzen, wie ein entstehendes System in ihre Arbeitsabläufe integrierbar ist.

Prototyping ist ein Prozeßmodell der Software-Entwicklung, das speziell auf die oben skizzierten Randbedingungen - unklare Aufgabenstellung und komplexe Entwurfsaufgaben - abgestimmt ist. Prototyping setzt voraus, daß die späteren Benutzer in den Entwurfsprozeß von Systemen einbezogen werden. Gerade bei der Entwicklung von Benutzungsoberflächen ist dies eine unabdingbare Voraussetzung, um tragfähige Ergebnisse zu erzielen.

In diesem Bericht untersuchen wir, inwieweit Werkzeuge und Umgebungen für den Entwurf von Benutzungsoberflächen geeignet sind, Prototyping-Prozeße zu unterstützen. Es geht also darum, Konzepte und Praxis des Prototyping und der dabei verwendeten Werkzeuge zu diskutieren. Damit wollen wir einen Beitrag zur Bewertung und Weiterentwicklung von grundsätzlichen Aspekten des Prototyping und keine aktuelle Werkzeugübersicht bieten. Entsprechend haben wir bei unseren Einschätzungen der Werkzeuge und ihres Einsatzes auch nur diejenigen Merkmale hervorgehoben, die relevant für die gesamte Werkzeugart sind.

Eine derartige Untersuchung muß Erfahrungen aus Entwicklungsprojekten ebenso berücksichtigen wie die Analyse der technischen Gegebenheiten der einzelnen Werkzeuge. Wir stützen uns aber nicht nur auf die Erfahrungen unserer Interview-partner aus verschiedenen Projekten, wir haben auch unsere persänlichen Erfahrungen aus vielen Jahren als Software-Entwickler in die Studie eingebracht.

Wir haben diesen Bericht folgendermaßen aufgebaut: Im zweiten Kapitel führen wir die Terminologie ein, die wir in diesem Bericht verwenden, und beschreiben den Stellenwert von Prototyping im Software-Entwicklungsprozeß. Weiterhin gehen wir speziell auf die verschiedenen Arten von Benutzungsoberflächen sowie auf das Oberflächen-Prototyping ein.

Kapitel 3 beschreibt Merkmale und Eigenschaften von Werkzeugen zum Prototyping von Oberflächen. Wir konzentrieren uns dabei besonders auf die Werkzeugeigen-schaften, die unserer Meinung nach wesentlich sind, wenn mit ihnen Oberflächen-prototypen erstellt und diese anschliessend für ein Zielsystem weiterentwickelt werden.

Kapitel 4 enthält eine Klassifikation verschiedener Werkzeuge. Im Zusammenhang mit Oberflächen-Prototyping unterscheiden wir vier Gruppen: die Interface-Builder, die 4. Generationssysteme, Hypercard-basierte Werkzeuge und Objektorientierte Frameworks. Die verschiedenen Werkzeugarten werden anhand der in Kapitel 3 formulierten Eigenschaften vorgestellt. Für jede Werkzeugart werden ein bis zwei Werkzeuge vorgestellt, um die Konzepte zu verdeutlichen. Wir haben solche Werkzeuge als repräsentative Beispiele für die einzelnen Werkzeuggruppen ausgewählt, die auch in den in Kapitel 5 vorgestellten Projekten eingesetzt wurden. Dadurch wird es für den Leser einfachen, die geschilderten Fallstudien zu verstehen. Aufgrund der technischen Weiterentwicklung müssen diese Werkzeuge aber nicht mehr unbedingt den aller neuesten Stand der Technik repräsentieren. Dies schmälert aber ihren Wert in keiner Hinsicht, um die relevanten Konzepte zu verdeutlichen, da auf der Ebene der Werkzeugarten keine grundsätzlichen Weiterentwicklungen stattgefunden haben.

Kapitel 5 enthält eine umfangreiche Sammlung von industriellen Fallstudien, die nach der im Kapitel 4 eingeführten Werkzeugklassifikation strukturiert ist. In den vorgestellten Projekten wurden verschiedene Ausprägungen der Werkzeugarten eingesetzt, um Oberflächen-Prototypen zu konstruieren. Wir berichten, welche Werkzeuge wie eingesetzt wurden, welche Ergebnisse erzielt wurden und welche Erfahrungen die Entwickler dabei gemacht haben.

Im Kapitel 6 formulieren wir auf der Basis der vorgestellten Fallstudien unsere Analysen und bewerten den Einsatz von Werkzeugen beim Oberflächen-Prototyping. Weiterhin beschreiben wir Trends auf diesem Gebiet, die sich unserer Meinung nach abzeichnen.

Dieser Bericht dokumentiert die Ergebnisse der Arbeit des GI-AK 4.3.2 "Prototyping". Dieser Arbeitskreis hat in den letzten vier Jahren den Einsatz und die Akzeptanz von Prototyping in der industriellen Software-Entwicklung untersucht.

Die vorliegende Studie, die speziell das Oberfächen-Prototyping betrachtet, ist die zweite dieser Art. Die erste Studie, die allgemeine Prototyping-Projekte vorstellt und analysiert, ist als GMD-Studie erschienen [Kieback 91]. Die wesentlichen Ergebnisse sind in einem Beitrag des Informatik-Spektrums [Kieback 92] sowie im Tagungsband der "International Conference on Software Engineering 93" [Lichter 93] zusammenfassend dargestellt.

Abschließend soll darauf hingewiesen werden, daß dieser Bericht ohne die Unterstützung der einzelnen Projektteams, deren Erfahrungen beim Oberflächen-Prototyping in Kapitel 5 beschrieben sind, nicht zustande gekommen wäre. Im einzelnen danken wir dafür den beteiligten Mitarbeitern der im folgenden aufgeführten Unternehmen und Instituten:

IBM Entwicklungsgesellschaft, Bäblingen

Robert Bosch GmbH, Stuttgart

RWG GmbH, Stuttgart

Schweizerische Bankgesellschaft, Zürich

Siemens AG, München

Voest Alpine, Linz

Weiterhin danken wir Josef Bäsze (SBG Zürich) und Dmitri Hartmann (PSI, Berlin). Sie haben die Entstehung dieses Berichts durch Beiträge und kritische Anmerkungen unterstützt.

Kapitel 2

Grundlagen

In diesem Kapitel beschreiben wir die Grundlagen, die nach unserer Meinung für das Verständnis des Berichtes benätigt werden.

Wir führen zunächst unsere Terminologie im Bereich Prototyping ein und erläutern die wesentlichen Konzepte und Ziele. Anschließend beschreiben wir, die Arten von Benutzungsoberflächen, die heute vorwiegend in Anwendungen vorgefunden werden. Im letzten Teil dieses Kapitels haben wir unsere Auffassung vom Prototyping von Benutzungsoberflächen formuliert.

2.1 Methodische Grundbegriffe

Obwohl die Diskussion zum Thema Prototyping schon seit mehr als 10 Jahren mehr oder minder intensiv geführt wird, kännen wir einige konzeptionelle oder methodische Besonderheiten dieser Diskussion feststellen:

Da ist zunächst ein deutlicher Unterschied zwischen der europäischen und der u.s.-amerikanischen Interpretation der Prototyping-Idee. In Europa ist es kaum noch umstritten, daß Prototyping ein adäquates und gezielt eingesetzes Mittel zur Kommunikation und Kooperation zwischen Entwicklern und den anderen beteiligten Gruppen – vor allem den Benutzern – darstellt. Dagegen gehen die Interessen in den Vereinigten Staaten in eine ganz andere Richtung, die auch durch den dort häufig verwendeten Begriff "Rapid Prototyping" angedeutet wird: die mäglichst rasche Konstruktion einer lauffähigen Softwareversion. Dies hat allerdings zur Folge, daß die Nicht-Entwickler nur selten in den Rückkopplungsprozeß mit einbezogen werden. Hinter beiden Sichtweisen steht sicherlich das Bestreben nach mehr Softwarequalität. Unterschiedlich sind jedoch die mit Softwarequalität verbundenen Merkmale. In Europa stellen wir fest, daß Software nicht mehr nur nach den traditionellen quantitativen Merkmalen wie Ablaufeffizienz, Speicherverbrauch und Fehlerfreiheit beurteilt werden. Eher wird die Dualität von anwendungsbezogener Funktionalität und benutzungsorientierter Handhabbarkeit in den Vordergrund der Qualitätsdiskussion gerückt. In den Staaten sehen wir dagegen eine Renaissance mäglichst quantitativer Ansätze, die sich auf die unmittelbare Messung von Softwarequalität oder der Qualität des Konstruktionsprozesses beziehen.

Auch in der europäischen Diskussion um Prototyping lassen sich Tendenzen feststellen. Das traditionelle Software Engineering und die an formalen Methoden orientierten Ansätze stehen der Idee nach wie vor mißtrauisch gegenüber. Mit dem immer noch vorherrschenden Bild von der Informatik als einer der Mathematik und

damit der formalen Analyse verpflichteten Disziplin, scheint sich der an Experiment und Erfahrung orientierte Ansatz des Prototyping schlecht zu vertragen. Nur so kännen wir uns erklären, daß die methodische und disziplinierte Verwendung von Prototyping-Konzepten, so wie wir sie auch hier vorstellen, oft ignoriert oder mißinterpretiert wird.

2.1.1 Prototyping und Prototyp

Was verstehen wir unter den Begriffen:

Prototyping ist die spezielle Ausprägung des Prozesses der Systementwicklung, bei dem Prototypen entworfen, konstruiert, bewertet und revidiert werden. Ziele des Prototyping sind:

- eine Kommunikationsbasis für alle beteiligten Gruppen, insbesondere zwischen Benutzern und Entwicklern zu schaffen, um zu einer gemeinsamen Projektsprache zu gelangen,
- durch Experimente und praktischen Umgang Erfahrungswissen über die Konstruktion und den mäglichen Einsatz des Softwaresystems zu schaffen,
- eine dynamische Beschreibung des angestrebten Systems zu erhalten und den sich ändernden Bedürfnissen und Einsichten anzupassen.

Ein *Prototyp* ist eine spezielle Ausprägung eines ablauffähigen Softwaresystems. Er realisiert ausgewählte Aspekte des *Zielsystems*, d.h. des zukünftigen Softwaresystems im Anwendungsbereich. Das bedeutet:

- Ein Prototyp ist *immer ablauffähig*. Eine reine Bildschirmmaske, die mit einem Grafikeditor erstellt wurde, sollte dementsprechend nicht als Prototyp bezeichnet werden.
- Ein Prototyp realisiert *Aspekte* des Zielsystems. Welche Aspekte in einem Prototyp umgesetzt werden, richtet sich nach den jeweils im Entwicklungsprozeß anstehenden Fragen. Zumeist, insbesondere bei interaktiven Anwendungen, werden sich diese Fragen auch auf die *Benutzungsoberfläche,* kurz die *Oberfläche* des Systems beziehen. Allerdings ist das nicht zwingend. Ein Prototyp kann sehr wohl modellieren, wie eine Anwendung mit einer Datenbank zusammenarbeitet, wobei von der Darstellung der späteren Oberfläche weitgehend abgesehen wird. Derartige Prototypen sind im Entwicklungsprozeß wichtig, stehen aber in dieser Studie nicht weiter zur Diskussion.
- Prototypen repräsentieren *ausgewählte* Aspekte, d.h. diese Aspekte müssen schon bei der Entwicklung des Prototyps festgelegt werden. Dies ist wichtig, weil die Auswahl bestimmt, welche Fragen von einem Prototyp beantwortet werden kännen und insbesondere, welche nicht sinnvoll an ihn zu stellen sind.

2.1.2 Die beteiligten Gruppen

Im Prototyping-Prozeß sind unterschiedliche Personengruppen mit verschiedenen Verantwortlichkeiten beteiligt. Da wir auf diese Gruppen an anderer Stelle [Kieback 92] genauer eingegangen sind, hier nur eine kurze Begriffsklärung:

- *Auftraggeber* und *Softwarehersteller* sind Vertragspartner eines Entwicklungsprojektes, auch wenn dies innerhalb einer Organisation abgewickelt wird.
- Die *Entwickler*, d.h. alle Mitglieder des Entwicklerteams, ungeachtet ihrer Position als Analytiker, Designer oder Programmierer, repräsentieren das softwaretechnische Wissen; die Verantwortung über ein Entwicklungsprojekt trägt das *DV-Management*. Die *Anwender*, die später das System direkt und indirekt einsetzen, repräsentieren das Fachwissen. Auf der fachlichen Seite macht es Sinn, noch das *Anwendermanagement* und die Gruppe der unmittelbaren *Benutzer* des Systems zu unterscheiden.

2.1.3 Kontext des Prototyping

Prototyping ist, wie gesagt, ein *Prozeß*, in dem "Produkte", also Prototypen und das Zielsystem, entwickelt werden. Damit ist der Systementwicklungsprozeß aber noch nicht insgesamt bestimmt. Offen bleibt, *wann welche* Prototypen zu *welchem Zweck* gebaut werden und *von wem* sie wie bewertet werden. Neben der Art und Weise, wie Software entwickelt wird, müssen noch weitere analytische, organisatorische und dokumentierende Aktivitäten festgelegt werden. Betrachten wir diese Festlegungen auf der konzeptionellen Ebene, dann sprechen wir von einer *Entwicklungsstrategie*. Prototyping ist in diesem Sinne immer *Teil* einer Entwicklungsstrategie. Da Prototyping Rückkopplungsprozesse explizit vorsieht und in diesen Rückkopplungsprozeß bewußt auch die Gruppe der Benutzer einbezieht, ist die gewählte Entwicklungsstrategie nicht unabhängig von Prototyping. Obwohl versucht wird [Kieback 92], den Bau von Prototypen in ein konventionelles Phasenmodell zu integrieren, hat sich diese Einbettung von Prototyping als problematisch erwiesen (vergl. [Lübbecke 93]). Vorteilhaft erweisen sich dagegen sog. evolutionäre Entwicklungsstrategien (vergl. [Budde 92]), bei denen sich entwerfende, konstruierende und bewertende Aktivitäten zyklisch abwechseln. Wir halten fest, daß Prototyping je nach Entwicklungsstrategie gefärdert oder in seinen Wirkmäglichkeiten behindert werden kann.

Wenn wir Prototypen unter statischen Aspekten betrachten, sprechen wir von ihrer *Architektur*. Ist eine solche Architektur für die Entwickler strukturell unverständlich, dann ist der Prototyp nicht weiterentwickelbar. Jeder revidierte Prototyp muß dann "von vorne" entwickelt werden. Wir werden mit Blick auf dieses Problem Anforderungen an die verwendeten Werkzeuge ableiten.

Damit sind wir beim Thema der *Entwicklungswerkzeuge*, zu denen wir auch entsprechende Sprachen rechnen. "Hand-codierte" Prototypen sind mit vertretbarem Aufwand nur in einer interpretativen Hochsprache sinnvoll, d.h. einer Sprache, die die unmittelbare Ausführung kleiner Konstruktionseinheiten (Prozeduren,

Funktionen) erlaubt. Die Konstruktion einer interaktiven grafischen Oberfläche muß auf jeden Fall durch vorgefertigte Bausteine und Anschlüsse an bestehende Fenstersysteme unterstützt werden. Steht für das Prototyping eine Entwicklungsumgebung zur Verfügung, mit der interaktive Software schnell durch Generatoren und Bausteinsammlungen konstruiert werden kann, so kommt der Auswahl der Zielsprache nicht mehr die vorrangige Bedeutung zu, da diese oft von der Entwicklungsumgebung generiert wird.

Bezogen auf unser Thema bleibt die Frage nach dem Leistungsumfang und der Beherrschbarkeit der vorgefertigten interaktiven Komponenten. Wir untersuchen entsprechend, in welchem Umfang Benutzungsoberflächen flexibel gestaltet werden kännen und welche vorgefertigten Bausteine die Werkzeuge für die Oberflächenentwicklung anbieten. Eng damit verbunden ist die Frage nach dem Aufwand, mit dem eine Benutzungsoberfläche entwickelt werden kann, und wie aufwendig es ist, diese zu modifizieren. Und schließlich betrachten wir, wie eine Oberfläche mit den anderen Teilen eines Softwaresystems verbunden werden kann.

Mit Blick auf die von uns für sinnvoll gehaltenen zyklischen Entwicklungsprozesse kännen für die jeweiligen Werkzeuge zum Bau der Oberflächen weitere Anforderungen an die resultierende Architektur von Prototypen gestellt werden. Dies bedeutet folgendes:

Beim Bau funktionaler Prototypen werden Oberflächenkomponenten mit den funktionalen Teilen des Systems verbunden. Als Konsequenz der Auswertungsprozesse werden dann sowohl die funktionalen als auch die Oberflächenkomponenten eines Prototyps weiterentwickelt. Welche konstruktiven Auswirkungen diese Weiterentwicklung hat, wird durch die Art des Werkzeugs und die Architektur des Prototyps bestimmt. Sind die funktionalen Teile z.B. als Zusätze in ein generiertes Rahmenwerk für die Interaktion eingefügt, dann führen €nderungen an der Benutzungsoberfläche meist dazu, daß die funktionalen Codesegmente "per Hand" in das neu generierte Gerüst übertragen werden müssen. Einfacher ist die Weiterentwicklung, wenn das Werkzeug die interaktiven Oberflächenkomponenten der Prototypen so kapselt, daß sie eine definierte Schnittstelle zur den ebenfalls gekapselten fachlichen Komponente besitzen (siehe dazu auch 2.3.1 "Architektur von Oberflächen").

2.1.4 Arten des Prototyping

In der europäischen Diskussion über Prototyping hat sich die von [Floyd 84] vorgeschlagene Begrifflichkeit der drei "E" durchgesetzt. Differenzierungen ergeben sich bei der Interpretation dessen, was unter *explorativ*, *experimentell* und *evolutionär* verstanden wird. Wir stellen zwei Richtungen fest: Die Sichtweise, die den *Prozeß* und die dabei angestrebten Ziele betrachtet (vergl. [Budde 92]) und die Sicht, die den Prototyp als *Produkt* in den Mittelpunkt stellt (vergl. [Bischofberger 92]).

Wir charakterisieren im folgenden beide Sichten durch Gegenüberstellung:

- *Exploratives Prototyping* (prozeßbezogen):
  Es soll die *Problemstellung klären*. Diskutiert werden veränderte Arbeitsinhalte und die Mäglichkeiten der DV-Unterstützung. Dazu werden ggf. alternative Prototypen konstruiert. Als Prototypart kommen meist Demonstrationsprototypen und funktionale Prototypen in Frage. Exploratives Prototyping eignet sich am ehesten zur Integration in ein konventionelles Vorgehensmodell. Es wird eingesetzt, um die Anforderungsanalyse zu unterstützen.
- *Exploratives Prototyping* (produktbezogen):
  Es betrachtet die *Bewertung* der Prototypen durch die *Anwender* und Benutzer. Damit ist so etwas wie die "Außensicht" auf den Prototyp charakterisiert. Gegenstand der Betrachtung ist die Art und Weise, wie die verschiedenen Prototyparten von den Anwendern fachlich bewertet werden (kännen). Damit steht als Fragestellung die fachliche Ausrichtung und die Handhabbarkeit im Vordergrund. Exploratives Prototyping ist nicht auf eine bestimmte Prototypart eingeschränkt.
- *Experimentelles Prototyping* (prozeßbezogen):
  Es klärt die *technische Umsetzung* der Anforderungen an ein System. Dabei konkretisieren die Benutzer experimentell ihre Vorstellungen von der DV-Läsung, und die Entwickler sammeln Erfahrungen über Machbarkeit und Zweckmäßigkeit eines Anwendungssystems. Dies geschieht meist anhand von funktionalen Prototypen und Labormustern. Beim experimentellen Prototyping läßt sich eine klassische Phasentrennung kaum noch aufrecht erhalten.
- *Experimentelles Prototyping* (produktbezogen):
  Es betrachtet die *Konstruktion* und *Bewertung* der Prototypen durch die *Entwickler*. Durch experimentelles Prototyping sollen die Machbarkeit einer DV-Läsung geklärt und über Designalternativen entschieden werden. Da diese Aussagen auch auf das Zielsystem zutreffen müssen, sind funktionale Prototypen und Labormuster der Gegenstand des experimentellen Prototyping. Dabei ist es nicht vorrangig von Interesse, ob die Anwender in den Prozeß integriert sind.
- *Evolutionäres Prototyping* (prozeßbezogen):
  Es ist Teil eines *kontinuierlichen Verfahrens*, in dem ein Anwendungssystem innerhalb einer Organisation an die sich ändernden Randbedingungen angepaßt wird. Dabei wird Prototyping nicht mehr in einem einzelnen in sich geschlossenen Projekt eingesetzt, sondern dient zur Unterstützung der kontinuierlichen Weiterentwicklung der DV-technischen Infrastruktur einer Organisation. Beim evolutionären Prototyping kann das ganze Spektrum der Prototyparten zum Einsatz kommen. Große Bedeutung haben Pilotsysteme, da selten Zielsysteme im traditionellen Sinne entwickelt werden.

- *Evolutionäres Prototyping* (produktbezogen):
  Es bedeutet die *kontinuierliche Weiterentwicklung* eines Prototyps bis zum Zielsystem. Je nach gewählter Strategie kann dies den horizontalen oder vertikalen Ausbau oder die Erweiterung von relativ selbständigen Prototyp-komponenten in einem Anwendungsrahmen bedeuten. Evolutionäres Prototyping wird meist funktionale Prototypen und Pilotsysteme zum Gegenstand haben.

Wenn wir in dieser Studie die Begriffe *explorativ*, *experimentell* und *evolutionär* ohne weitere Qualifikation gebrauchen, dann beziehen wir uns auf die prozeßorientierte Interpretation; wollen wir die produktbezogene Sicht in den Mittelpunkt stellen, machen wir dies ausdrücklich klar.

In der Praxis der Software-Entwicklung finden wir spezifische Ausprägungen dieser Arten des Prototyping je nachdem, ob ein Standardprodukt für einen mehr oder minder anonymen Markt oder ob ein zugeschnittenes Anwendungssystem entwickelt werden soll.

2.1.5 Arten von Prototypen

Während die *Arten des Prototyping* sich auf *Fragestellungen* beziehen, die im Prototyping-Prozeß aufkommen, richtet sich die Einteilung nach *Prototyparten* auf die *Gestaltung* eines *Prototyps* im Sinne eines Produktes dieses Entwicklungsprozesses. Wir unterscheiden folgende Arten von Prototypen:

- Ein *Demonstrationsprototyp* zeigt nur die prinzipiellen Einsatzmäglichkeiten, meist nur die mäglichen Handhabungsformen des künftigen Systems. Demonstrationsprototypen dienen besonders in der Start- oder Akquisitionsphase eines Projektes, um Entscheidungen vorzubereiten und zu helfen, eine Vision über das zukünftige System zu entwickeln. Sie sind daher "Wegwerfprodukte", die schnell und einfach gebaut werden kännen. Ihre fachliche "Detailtreue" oder ihr software-technischer Standard ist nachrangig.
- Funktionale *Prototypen*, also Prototypen im eigentlichen Sinne, modellieren in der Regel Ausschnitte der Benutzungsoberfläche und Teile der Funktionalität. Sie sind entlang einer "horizontalen" und einer "vertikalen" Dimension konstruiert. Als horizontale Dimension wollen wir in diesem Zusammenhang die Benutzungsoberfläche bezeichnen; die vertikale Dimension bezieht sich auf die "Tiefe" der Implementierung, ausgehend von der Oberfläche bis zur untersten Schicht des Zielsystems. Diese Prototypen kännen in ihrer Architektur bereits dem Zielsystem entsprechen.
- Ein *Labormuster* modelliert einen technischen Aspekt des späteren Anwendungssystem. Dieser Aspekt kann sich je nach Fragestellung auf die Architektur oder auf die Funktionalität beziehen. Labormuster sind für die Entwickler ein Experimentalsystem und eine Form von Machbarkeitsstudie. Sie müssen daher nicht zwingend in das Zielsystem eingehen.
- Ein *Pilotsystem* ist ein Prototyp von solcher Ausbaustufe und "Reife", daß er im Anwendungsbereich und nicht nur unter Laborbedingungen eingesetzt werden

kann. Er realisiert innerhalb eines Entwicklungsrahmens einen abgeschlossenen Teil des Zielsystems und wird schrittweise ausgebaut. Auch seine komfortable und sichere Bedienbarkeit und ein Mindestmaß an Benutzungsdokumentation unterscheiden ihn qualitativ von anderen Prototypen.

In dieser Studie sprechen wir sehr oft von *Oberflächenprototypen.* Damit ist keine weitere Art von Prototyp gemeint, sondern wir betrachten solche Demonstrationsprototypen, funktionalen Prototypen und Labormuster, bei denen die Modellierung der Benutzungsoberfläche im Vordergrund steht. Inwieweit dazu fachliche Funktionalität kommt, richtet sich nach der jeweiligen Prototypart. Bei Pilotsystemen gehen wir davon aus, daß dort Oberfläche und fachliche Funktionalität gleichermaßen vorhanden sind.

2.2 Benutzungsoberflächenarten

Zur groben Klassifikation von Benutzungsoberflächen unterscheiden wir drei Gruppen:

1. *Zeichenorientierte Benutzungsoberflächen*
bieten meist keine oder nur sehr eingeschränkte Mäglichkeiten, um Grafik darzustellen. Ihr wichtigster Vorteil ist, daß sie auf sehr vielen Hardware-Plattformen eingesetzt werden kännen, da sie nur geringe Anforderungen an die Bildschirmgeräte stellen. Andererseits sind die mäglichen Benutzungsoberflächen natürlich sehr einfach und für viele Anwendungen (z.B. Visualisierung von Prozessen) nicht ausreichend. Um zeichenorientierte Oberflächen zu erstellen, werden schon seit längerer Zeit Werkzeuge wie die sog. Maskengeneratoren eingesetzt.

2. *Grafische Benutzungsoberflächen*
ermäglichen die Realisierung beliebiger Grafiken auf Bitmap-Displays. Auf diesen Bildschirmen werden nicht mehr ganze Zeichen, sondern einzelne Bildpunkte (Pixel) angesprochen. Derartige Benutzungsoberflächen sind die Voraussetzung für sogenannte WYSIWYG-Systeme (What you see is what you get) und für die ganze Klasse der WIMP-Oberflächen (Window, Icons, Menus, Pointers). Für ihre Gestaltung gibt es verschiedene Arten von Werkzeugen wie Interface-Builder, Application Frameworks etc. Das aus unserer Sicht charakteristische Merkmal ist, daß grafische Benutzungsoberflächen auf die Darstellung am Bildschirm festgelegt sind. Neben den Bitmap-basierten Systemen gibt es für Anwendungen besonders im CAD-Bereich sog. Vektorgrafik-Systeme, die wir aber hier nicht weiter betrachten.

3. *Multimodale Benutzungsoberflächen*
erlauben, daß die Benutzer mit dem System über verschiedenste Kommunikationskanäle interagieren und dabei mehr als nur grafische Bildschirmdarstellungen verwenden (z.B. Gestik, Sprache, Datenhandschuhe, Virtual Reality). Für diese Oberflächen gibt es noch keine kommerziell einsetzbaren Entwurfswerkzeuge. Die Gestaltung solcher Oberflächen ist derzeit noch weitgehend ein Forschungs- und Entwicklungsthema.

Für den Kontext dieser Studie sind also primär grafische Benutzungsoberflächen interessant. Uns ist klar, daß in der industriellen Praxis vielfach noch zeichenorientierte Oberflächen, die mit Maskengeneratoren erstellt werden, überwiegen. In [Kieback 92] wird z.B. über ein Prototyping-Projekt berichtet, in dem eine große Anzahl von Masken für eine zeichenorientierte Benutzungsoberfläche erstellt wurde. Allerdings kann diese Technologie heute als überholt bezeichnet werden und wird tendenziell auch stark an Bedeutung verlieren. Eine detaillierte Einschätzung findet sich in [Budde 92].

2.3 Oberflächen-Prototyping

Bei der Arbeit an dieser Studie über Oberflächen-Prototyping haben wir festgestellt, daß uns eine einheitliche Terminologie fehlt. Jeder, der in einer Anwendungs- oder Entwicklungswelt zu Hause ist, verwendet mitunter sehr unterschiedliche Begriffe, teils deutschsprachige, teils als Anglizismen. Wir haben deshalb in den folgenden Abschnitten die Terminologie im Bereich Oberflächen-Prototyping erläutert, die Grundlage dieses Berichts ist.

2.3.1 Begriffe

Oberflächenelemente

Eine Oberfläche ist mehr als das Bildschirm-Layout. Oberflächen zeigen natürlich zunächst bestimmte grafische *Darstellungsformen*, also das, was auf dem Bildschirm unmittelbar sichtbar ist. Heute sind diese Darstellungsformen nicht mehr beliebig, sondern lassen sich meist derjenigen Fenstersystemfamilie zuordnen, mit deren Hilfe sie realisiert worden sind. Oft spricht man in diesem Zusammenhang auch von dem charakteristischen "Look" einer Oberfläche.

Oberflächen sind aber nicht nur durch statisch angeordnete grafische Elemente charakterisiert, sondern auch durch die Art und Weise, wie sich diese Elemente verhalten und wie sie unter Verwendung der verfügbaren Eingabegeräte *gehandhabt* werden kännen. Wenn Benutzer die Elemente einer Oberfläche interaktiv verändern oder beeinflussen kännen, sprechen wir von den *Umgangsformen*, die eine Oberfläche ermäglicht. Dieses wird häufig auch als das "Feel" einer Oberfläche bezeichnet. Viele Oberflächen, z.B. zur Steuerung technischer Systeme, enthalten aber auch Elemente, die sich nicht durch Benutzeraktionen, sondern durch Anwendungsprozesse verändern. Sie zeigen in diesem Fall ein interaktives *Verhalten*.

Bei der Gestaltung einer interaktiven Oberfläche hängen Darstellungsformen und Umgangsformen oder Verhalten immer eng zusammen. Wir sprechen von einem *Oberflächenelement*, wenn wir die kleinste im Entwurf und in der Anwendung verfügbare Einheit meinen. Ein Knopf, ein Auswahlmenü oder ein Fenster mit Rollbalken sind solche Oberflächenelemente. Gelegentlich ist es sinnvoll, von *Interaktionselementen*, als den interaktiven Einheiten und von *passiven grafischen Elementen* oder von *Dekorationen* als den nicht-interaktiven Elementen zu sprechen. In diesem Bericht behandeln wir aber vorrangig Interaktionselemente wie eine Menüauswahl, die als Umgangsform eine entsprechende Handhabung mit Maus

oder Tastatur bietet, aber auch eine charakteristische Darstellungs- und Rückkopplungsstrategie hat (z.B. Invertierung eines mit Mausklick angewählten Textelements in diesem Auswahlmenü). Wieder bezogen auf Fenstersysteme wie Motif oder Windows spricht man häufig vom *Look & Feel*, wenn man charakterische Darstellungs- und Umgangsformen auf die einzelnen Oberflächenelemente bezieht.

Oberflächenelementarten

Im Entwurfsprozeß, also bei der Anordnung der Oberflächenelemente zu einem Layout, entwickeln wir nicht jedes einzelene Elemente von neuem, sondern greifen auf einen vorgegebenen Satz solcher Elemente zurück. Die Mengen gleichartiger Elemente bezeichnen wir als Oberflächenelementarten oder entsprechend Interaktionselementarten. Man kännte sie auch Elementtypen nennen. Allerdings kollidiert das gelegentlich mit dem Verständnis des Typbegriffs in Programmiersprachen.

Entscheidend für die Gestaltung von Oberflächen ist, welche Oberflächenelementarten vom jeweiligen Entwicklungswerkzeug angeboten werden und mit welchem Aufwand dieser Vorrat vergräßert werden kann.

Architektur von Oberflächen

Wenn Oberflächen also mehr sind als ihre äußere (statische) Darstellung, wenn Verhalten dazukommt und wenn schließlich die Oberfläche in Verbindung mit einer fachlichen Anwendung gebracht werden muß, dann schlägt sich das in unterschiedlichen Architekturen von Oberflächenprototypen nieder. Wir unterscheiden zwei konzeptionelle Komponenten, auf die wir uns im weiteren immer wieder beziehen: Es bietet sich an, alle Bestandteile einer Anwendung oder eines separierbaren Teils einer Anwendung, die die Oberfläche mit ihren Darstellungsformen und ihrem interaktiven Verhalten realisiert, in einer Komponente zusammenzufassen – wir nennen sie *Oberflächenkomponente*. Davon getrennt gruppieren wir die fachlichen Anteile der Anwendung in der *fachlichen Komponente*. Für einige Formen von Oberflächen-Prototyping ist es wichtig, daß sowohl die Oberflächen als auch fachlichen Komponenten mit einer *Datenbankkomponente* interagieren. Wir halten fest, daß dies zunächst nur eine konzeptionelle Einteilung ist. Ein wesentlicher Gesichtspunkt in der weiteren Diskussion wird sein, inwieweit sich besonders die Oberflächenkomponente und die fachliche Komponente auch software-technisch getrennt in eigenen Modulen, Klassen oder Subsystemen fassen lassen.

Interaktive Anwendungssysteme

Mit der rasch wachsenden Verbreitung grafischer Arbeitsplatzrechner kommt der Oberflächengestaltung und damit auch dem Oberflächen-Prototyping zentrale Bedeutung zu. Hier sind es – wie gesagt – die interaktiven Arbeitsplatzsysteme, oft im Zusammenspiel mit Serverrechnern, die nach dem Leitbild des elektronischen Schreibtischs für Büroanwendungen im Mittelpunkt stehen. Aber auch im Produktionsbereich werden heute bei Echtzeitsystemen für Prozeßsteuerung und -kontrolle wachsende Anforderungen an die Oberflächengestaltung gestellt. Deshalb

soll dieser Bereich in seiner Bedeutung für das Oberflächen-Prototyping ebenfalls diskutiert werden. Oberflächen-Prototyping, so wie wir es verstehen, bezieht sich auf interaktive Anwendungssysteme, die auf grafischen Arbeitsplatzrechnern oder vergleichbaren Rechnern eingesetzt werden. Meist sind diese Rechner in Netze eingebunden. Natürlich sind auch die im Großrechnerbereich auf Datenbanken arbeitenden Informationssysteme zu den interaktiven Anwendungen zu rechnen. Wir haben schon darauf hingewiesen, daß sich dabei die Oberfläche auf Bildschirmmasken mit Cursor- oder Funktionstastensteuerung reduziert. Ebenso haben die auf Spezialhardware im Großrechnerbereich laufenden "traditionellen" CAD/CAM-Anwendungen für unsere Betrachtung geringere Bedeutung, da hier die Mäglichkeiten des Prototyping sehr begrenzt sind und im Regelfall "Handarbeit" bei der Oberflächengestaltung vorherrscht.

Reaktive Systeme

Ein wesentliches Merkmal der von uns betrachteten interaktiven Anwendungssysteme ist, daß sie (zumindest was ihr Verhalten an der Oberfläche anbetrifft) als *reaktive Systeme* bezeichnet werden kännen. In reaktiven Systemen geht jede *Aktivität* vom Benutzer aus. †ber die entsprechenden Eingabegeräte wird eine *Aktion* des Benutzers in ein *Ereignis* des Systems umgesetzt. Dieses Ereignis (oft event genannt) wird vom Fenstersystem einem als *aktiv* bezeichneten Ereigniskontext zugewiesen. Solche Ereigniskontexte kännen Fensterflächen oder, wie heute üblich, einzelne der bereits genannten Oberflächenelemente sein. Ein typisches Ereignis ist beispielsweise ein Mausklick über einem Oberflächenelement "OK-Knopf". Das Ereignis "Mausklick" wird also dem Oberflächenelement mitgeteilt. Kann das Oberflächenelement dieses Ereignis interpretieren, d.h. gibt ein zu diesem Ereignis passendes Programmstück zur Ereignisbehandlung (event handler), dann reagiert das System. In unserem Beispiel kännte der OK-Knopf Teil einer Dialogbox sein, in der der Benutzer zuvor bestimmte Einstellungen vorgenommen hat. Im Zusammenspiel von Oberflächenkomponente und fachlicher Komponente *reagiert* das Anwendungssystem auf dieses Ereignis, d.h. es wird zunächst fachlich geprüft, ob die gewählten Einstellungen zulässig und umsetzbar sind. Als interaktive Reaktion für den Benutzer zeigt die Oberfläche dann ggf. eine veränderte Darstellung, die dem Benutzer als Rückkopplung für weitere Aktivitäten dient. Dieser Zyklus aus Aktion und Reaktion, vermittelt über Ereignisse, wird auch *Interaktion* genannt.

2.3.2 Angrenzende Themen

Wir betrachten Oberflächen-Prototyping im Kontext einiger Themen:

Objektorientierung

Oberflächen-Prototyping bedeutet unter den skizzierten technischen Randbedingungen, daß die Anforderungen an die verwendete Architektur in vollem Umfang nur von objektorientierten Systemen erfüllt werden kännen. Wir kennen kein erfolgreiches Fenstersystem auf Arbeitsplatzrechnern, das nicht zumindest in seiner Konzeption objektorientiert aufgebaut ist. Wieweit sich dieses Prinzip nach außen in der Verwendung oder beim Ausbau eines Prototyps zeigt, ist heute noch sehr unterschiedlich. Hier werden oft hybride Läsungen angeboten, die auch in Zukunft noch ihre Bedeutung bei der Integration interaktiver Anwendungen in bestehende Systeme haben werden. Wir werden Anforderungen an die Architektur von Oberflächenprototypen diskutieren, und die jeweiligen Konsequenzen für den Entwicklungsprozeß aufzeigen.

Oberflächenprototyp und Zielsystem

In diesem Zusammenhang stellt sich allgemein die Frage nach dem Verhältnis von Oberflächenprototyp und Zielsystem. Aus dem bisher Gesagten läßt sich ableiten, daß ein evolutionärer Ansatz, d.h. der schrittweise Ausbau eines Prototyps zum eingesetzten Zielsystem, die erwünschte Variante ist. Alle konzeptionellen oder architektonischen Brüche, die sich beim Wechsel der Implementationsplattform, der Zielsprache oder sogar des Fenstersystems ergeben, machen Abstriche an den Aussagen und Vorgaben, die durch das Prototyping gewonnen wurden.

Benutzerbeteiligung

Oberflächen-Prototyping bedeutet, daß die Benutzer in den Entwicklungsprozeß integriert werden. Wenn schon Prototyping als Prinzip aus unserer Sicht eng mit der Bewertung der Prototypen durch die späteren Benutzer verknüpft, so macht Oberflächen-Prototyping erst recht wenig Sinn, wenn die Benutzer nicht in den Prozeß einbezogen sind. Diese Einsicht stäßt innerhalb von Entwicklerorganisationen bisweilen noch auf Widerstände.

Entwicklung durch Benutzer

In eine ganz andere Richtung gehen demgegenüber Forderungen, Oberflächen für Prototypen oder gar das Zielsystem von den Benutzern selbst erstellen zu lassen. Zunächst sollte bei diesem Thema zwischen der Konfiguration und der parametrisierten Einstellung einer Benutzungsoberfläche und ihrem vollständigen Entwurf durch Benutzer unterschieden werden. Während ersteres auf eine bewährte Praxis zurückgreifen kann, sind der Benutzerprogrammierung enge technische und konzeptionelle Grenzen gesetzt. Wir gehen davon aus, daß der Entwurf und die Konstruktion komplexer, interaktiver Anwendungen auch noch in absehbarer Zukunft in der Zusammenarbeit von Anwendungsfachleuten und Software-Technikern geschieht.

Evolutionäre Systementwicklung

Wir haben gesagt, daß Prototyping in eine Entwicklungsstrategie eingebettet ist. Dabei ist es – wir haben auch darauf hingewiesen – durchaus denkbar, daß Prototyping als "Ausweitung" der Anforderungsanalyse verstanden wird. Aber je intensiver sich die Funktionalität einer Anwendung und ihre Oberflächengestaltung durchdringen, desto deutlicher wird, daß prototypische Entwicklungszyklen den gesamten Prozeß bestimmen müssen. Daher sollte Oberflächen-Prototyping vor dem Hintergrund der Konzepte evolutionärer Systementwicklung diskutiert werden.

Entwurfsmetaphern

In den letzten Jahren hat die Diskussion um Entwurfsmetaphern im Software Engineering eine wachsende Bedeutung gewonnen. Dahinter steht die Einsicht, daß Entwufsmethoden und Entwicklungswerkzeuge noch keine Anleitung geben, wie ein Anwendungssystem konkret gestaltet werden soll. Wir stellen immer wieder fest, daß den Entwicklern trotz methodischen Wissens und vorhandener Werkzeuge eine Orientierung fehlt, die ihnen eine Grundlage für Entwurfsentscheidungen und eine "Vision" des zukünftigen Systems gibt. Eine solche orientierende Vision für die Gestaltung eines Systems nennen wir Leitbild (z.B. "das papierlose Büro" oder "die gutsortierte Werkbank"). Ein Leitbild wird durch "plastische" bildhafte Entwurfsmetaphern (z.B. der "elektronische Karteikasten") erst wirklich umsetzbar. Untersuchungen [Maaß 92] haben gezeigt, daß in vielen (erfolgreichen) Software-Systemen Entwurfsmetaphern eingegangen sind. Wir wollen in diesem Bericht keine eigene Diskussion über Entwurfsmetaphern führen. Aber an den Stellen, wo es sich anbietet, werden wir darauf hinweisen, welche Entwurfsmetaphern hinter den von uns untersuchten Werkzeugen stehen. Denn diese Entwurfsmetaphern geben einen guten Eindruck vom Einsatzschwerpunkt eines Werkzeugs und sie helfen, geeignet mit den Werkzeugen umzugehen.

Kapitel 3

Werkzeuge zum Oberflächen–Prototyping

In diesem Kapitel diskutieren wir Eigenschaften von Werkzeugen, mit denen Benutzungsoberflächen gestaltet werden kännen. Diese Werkzeuge ermäglichen, die Oberflächenkomponente auf einem hohen Abstraktionsniveau mit der fachlichen Komponente der Anwendung zu verbinden. Bei der Auswahl der Merkmale haben wir besonders die Eignung für das Oberflächen-Prototyping berücksichtigt. Folgende Fragestellungen, die sich in den Merkmalen der Werkzeuge widerspiegeln, scheinen uns in diesem Zusammenhang relevant:

- Welche Mäglichkeiten bieten Werkzeuge, um beim Entwurf Oberflächenelemente anzuordnen?
- Welche Arten von Oberflächenelementen werden angeboten und welche Funktionalität bieten sie?
- Wie kännen die vorhandenen Oberflächenelementarten durch den Benutzer des Werkzeugs erweitert werden?
- Wie interagieren die Elemente einer Oberfläche?
- Welche Datenmodelle unterstützt das Werkzeug an der Schnittstelle zur Anwendung?
- Wie kännen erstellte Oberflächen dynamisch ausgeführt werden?
- Wie kann die fachliche Komponente der Anwendung an die Oberfläche angebunden werden?
- Kann ein Oberflächenprototyp für das Zielsystem weiterverwendet werden?
- Welchen Anteil hat die Entwicklungsumgebung an der Laufzeitumgebung für den Oberflächenprototyp?
- Kännen mit den Werkzeugen Oberflächen gestaltet werden, die vorgegebenen Richtlinien entsprechen?

Eigenschaften von Programmbibliotheken, mit denen ebenfalls Benutzungsoberflächen konstruiert werden kännen, werden in diesem Kapitel explizit nicht betrachtet. Da Oberflächen mit diesen Bibliotheken auf dem Niveau konventioneller imperativer Sprachen programmiert werden müssen, sind sie für das Oberflächen-Prototyping weniger geeignet.

3.1 Gestaltung der Oberflächen

Werkzeuge kännen danach klassifiziert werden, wie mit ihnen Benutzungsoberflächen definiert werden kännen. In [Myers 92b] werden zwei wesentliche Klassen unterschieden:

- *Direkt-manipulative Werkzeuge*
  erlauben, daß die angebotenen grafischen Elemente einer Oberfläche beim Entwurf durch direkte Interaktion, d.h. durch Auswahl und Positionierung (auch direkte Manipulation genannt) mit der Maus, zu einem Ganzen zusammengesetzt werden kännen.
- *Sprachbasierte Werkzeuge*
  stellen eine eigene Sprache zur Verfügung, mit der die Elemente einer Oberfläche und deren Positionen textuell, aber auf einem hohen Abstraktionsniveau, definiert werden kännen.

Es ist wesentlich komfortabler, einfacher und schneller, Benutzungsoberflächen direkt-manipulativ am Bildschirm zusammenzubauen als diese in einer eigenen Sprache zu spezifizieren.

Da die Geschwindigkeit, mit der eine Oberfläche definiert und weiterentwickelt werden kann, von großer Wichtigkeit für das Prototyping ist, sind grafische Werkzeuge den sprachbasierten Werkzeugen vorzuziehen. Im folgenden gehen wir ausschließlich auf solche Werkzeuge ein. Da das Spektrum der Funktionalität direkt-manipulativer Werkzeuge sehr breit ist, untersuchen wir im weiteren folgende Anforderungen:

- Der Entwickler muß unterstützt werden, wenn er Oberflächenelemente relativ zu anderen Elementen positionieren will.
- Da es oft mühsam ist, Oberflächenelemente absolut zu positionieren, muß das Werkzeug dies erleichtern. Sinnvolle Unterstützungsmechanismen sind:
  - Ein Raster, das verhindert, daß Oberflächenelemente gegeneinander nur um wenige Bildpunkte verschoben sind.
  - Explizites Gruppieren der Oberflächenelemente, wobei definiert werden kann, wie die Elemente untereinander ausgerichtet werden sollen.
  - Hierarchischer Aufbau der Oberflächenelemente, wobei das übergeordnete Element festlegt, wie die von ihm eingeschlossenen Elemente positioniert werden.
- Mit dem Werkzeug muß definiert werden kännen, wie sich die Oberflächenelemente verhalten sollen, wenn sich die Gräße des Fensters ändert, das die Benutzungsoberfläche begrenzt.

3.2 Funktionalität der Oberflächenelemente

Werkzeuge zum Prototyping von Oberflächen kännen dahingehend unterschieden werden, welche Arten von Oberflächenelementen vom Entwickler verwendet werden kännen, und ob die Menge der angebotenen Oberflächenelementarten erweitert werden kann.

Praktisch alle Werkzeuge bieten die Elementarten Textfeld, Texteditor, Menü und Druckknopf an. Komplexere Oberflächenelemente sind beispielsweise Matrizen, Tabellenkalkulationen, Diagramme und Datenbank-Browser.

Ein Schritt in Richtung Abstraktion bieten solche Systeme, bei denen nicht konkrete Oberflächenelemente, sondern abstrakte Interaktionstypen ausgewählt werden. D.h. im Entwurfsprozeß wird z.B. nicht ein bestimmtes Auswahlmenü mit seiner konkreten Darstellung beschrieben, sondern nur seine 1-aus-n-Auswahl mit den jeweiligen Auswahlelementen und eine relative Anordnung innerhalb des Layouts. Die konkrete Umsetzung auf eine bestimmte Oberfläche und ein Fenstersystem geschieht dann erst im konkreten Generierungsprozeß und kann über Parameter gesteuert werden. Auf diese Weise lassen sich aus einem Entwurf Oberflächenprototypen für unterschiedliche Systemplattformen realisieren.

3.3 Erweiterbarkeit der Oberflächenelementarten

Während die meisten Werkzeuge einen ähnlichen Grundvorrat von Oberflächenelementarten zur Verfügung stellen, ist die Flexibilität der Werkzeuge und die Qualität der angebotenen Elementarten durchaus unterschiedlich. Da ein Werkzeug nur die für sehr einfache oder sehr zugeschnittene Anwendungen benätigten Arten von Oberflächenelementen standardmäßig zur Verfügung stellen kann, ist es wichtig, daß die Palette der angebotenen Oberflächenelementarten einfach erweitert werden kann.

Grundsätzlich sehen wir drei Mäglichkeiten, die vorhandenen Elementarten zu erweitern:

- durch Programmierung einer neuen Elementart in einer konventionellen Sprache,
- durch Kombination vorhandener grafischer Elemente und der Erweiterung ihres Verhaltens durch Programmtexte in einer speziellen Sprache,
- durch Zukauf von auf dem Markt angebotenen Elementarten.

Unter den hier betrachteten Gesichtspunkten genügt es nicht, wenn ein Werkzeug eine prozedurale Schnittstelle bietet, damit neue Elementarten als Programme eingebracht werden kännen. Dieses Verfahren erfordert meist sehr genaue Kenntnisse der inneren Werkzeugstruktur.

Wesentlich einfacher ist es, wenn neue Elementarten mit vorhandenen Mechanismen in das grafische Entwurfswerkzeug integriert werden kännen. Das kann zum einen durch explizite Beschreibung einer neuen Elementart in einer speziell dafür geeigneten Sprache geschehen. Eine solche Beschreibung wird dann als Datei in ein vordefiniertes Verzeichnis gelegt. Noch einfacher ist es, wenn die gewünschte Oberflächenelementart interaktiv direkt mit dem Werkzeug aus vorhandenen Elementteilen kopiert oder mit einem speziellen Editor erstellt werden kann.

Eine wesentliche Unterstützung des Prototyping bieten Werkzeuge, für die neue Oberflächenelementarten auf dem Markt angeboten werden, die dann einfach eingebaut werden kännen. Dadurch wächst die Wahrscheinlichkeit, daß bei Bedarf auch relativ komplexe Elementarten in kurzer Zeit zur Verfügung stehen.

3.4 Kooperation zwischen den Oberflächenelementen

Um Oberflächenprototypen schnell zu erstellen, ist es wichtig, daß die Handhabung der Oberflächenelemente und deren Zusammenarbeit auf einfache Weise in der Oberflächenkomponente definiert werden kann, ohne daß dafür eine konventionelle Programmiersprache verwendet werden muß. Soll eine vollständige Anwendung aus Oberflächen- und fachlicher Komponente erstellt werden, ist dieses Kriterium hingegen nachrangig.

Es gibt die folgenden Ansätze, um die Handhabung einer Oberfläche auf einfache Art und Weise beschreiben zu kännen:

- Ein verbreiteter Ansatz ist die Verwendung spezieller, auf die Ereignisbehandlung zugeschnittener Programmiersprachen, oft *Skriptsprachen* genannt. Programmtexte der Skriptsprache werden den einzelnen Oberflächenelementen zugeordnet. Mit ihrer Hilfe läßt sich die Handhabung, die Weiterleitung von Ereignissen und oft auch die Darstellung der Oberflächenelemente beschreiben.
- Der Entwickler kann grafisch spezifizieren, welche Ereignisse wie von einem Oberflächenelement an ein anderes weitergeleitet werden sollen. Beispielsweise kann für einen Knopf spezifiziert werden, daß er einem Fenster die Nachricht "äffne-dich" schickt, wenn er gedrückt wird.
- Hypertext-Referenzen erlauben, daß aufgrund einer vordefinierten Verbindung von einem Kontext des Hypertexts in einen anderen gesprungen werden kann. Hypertext-basierte Werkzeuge eignen sich deshalb speziell, um reine Benutzungsoberflächen ohne fachliche Komponente zu modellieren.

3.5 Datenmodellunterstützung

Gute Werkzeuge zum Prototyping gibt es besonders in klar abgrenzbaren Anwendungsgebieten. Dort lassen sich Benutzungsoberflächen aus einem vordefinierten Satz von Oberflächenelementarten aufbauen. In solchen Anwendungsgebieten kann das gewünschte Anwendungsverhalten auf einem hohen Abstraktionsniveau definiert und sofort mit einem Prototyp experimentiert werden.

Ein typischer Fall für ein gut abgrenzbares Anwendungsgebiet, das über Benutzungsoberflächen hinausgeht, sind Informationssysteme, die auf Datenbanken basieren. Dort kommt es im wesentlichen auf das Eingeben, Verwalten und Sichten von strukturierten Informationsmengen an. Werkzeuge, die dieses Anwendungsfeld abdecken, kännen aufgrund des unterstützten Datenmodells klassifiziert werden. Die derzeit kommerziell erhältlichen Werkzeuge unterstützen eines der folgenden vier Datenmodelle:

- das relationale Datenmodell,
- das quasi-relationale Datenmodell,
- eine Datei mit Daten einheitlicher Struktur (Records),
- Stapel von einheitlich strukturierten Karten in Hypertext-Systemen.

Mittlerweile gibt es erste Werkzeuge, die das objektorientierte Datenmodell unterstützen.

Normalerweise wirkt sich das unterstützte Datenmodell wie folgt auf die Eignung eines Werkzeugs für das Prototyping aus:

- Werkzeuge, die ein relationales bzw. ein quasi-relationales Datenmodell unterstützen, eignen sich meist sowohl für exploratives als auch für evolutionäres Prototyping. Hier ist von Vorteil, daß eine formulorartige Darstellung an der Oberfläche sehr einfach auf das Datenmodell abgebildet werden kann und umgekehrt. Der hauptsächliche Unterschied zwischen den beiden Werkzeugarten ist, daß bei Werkzeugen für ein quasi-relationales Modell die Kopplung zwischen Datenhaltung und Benutzungsoberfläche wesentlich enger ist, als bei Werkzeugen für das rein relationale Modell. Quasi-relationale Werkzeuge kännen üblicherweise aus graphischen Entwürfen ein Datenmodell ableiten und daraus standardisierte Formulare für die Darstellung und Bearbeitung der Daten generieren. Werkzeuge, die ein rein relationales Modell unterstützen, kommunizieren meist über SQL mit einer Datenbank. Aufgrund kürzerer Entwicklungszeiten sind Werkzeuge für ein quasi-relationales Modell besser geeignet, um explorativ vorzugehen.
- Werkzeuge, die nur Dateien mit Daten einheitlicher Struktur unterstützen, sind aufgrund ihrer eingeschränkten Funktionalität kaum für allgemeine Prototyping-Zwecke einsetzbar. Für einfache Anwendungen werden zugeschnitte Werkzeuge angeboten.

- Hypertext-Systeme – auf der Basis von Stapeln einheitlich strukturierter Karten – bieten eine große Flexibilität bei der Datenmodellierung, da die einzelnen Karten auch über Stapelgrenzen hinaus durch Hypertext-Referenzen miteinander verbunden werden kännen. Dadurch ist es mäglich, auch komplexe Datenstrukturen zu verwalten, z.B. relationale Datenmodelle. Der Aufwand dazu ist aber wesentlich gräßer, als wenn Werkzeuge eingesetzt wird, die das relationale Modell direkt unterstützen.

3.6 Ausführbarkeit

Es gibt drei Arten, um Oberflächenprototypen auszuführen:

- Die Oberflächenkomponente und die fachliche Komponente sind jederzeit interpretativ ausführbar.
- Die Oberflächenkomponente ist jederzeit interpretativ ausführbar; die fachliche Komponente muß jedoch selbständig übersetzt und zur Oberflächenkomponente gebunden werden.
- Aus der Oberflächenbeschreibung muß Code generiert werden, der dann zusammen mit der fachlichen Komponente übersetzt und zu einem ausführbaren Programm gebunden wird.

Für exploratives Oberflächen-Prototyping ist die erste Ausführungsart am besten geeignet, da sie erlaubt, den Prototyp jederzeit ohne zeitraubende Vorbereitungen auszuführen.

3.7 Anbindung der Funktionalität

Es gibt eine große Vielfalt, wie die fachliche Komponente an einen Oberflächen-Prototyp angebunden werden kann. Die verschiedenen Anbindungsarten lassen sich grob in die folgenden Kategorien einteilen:

- Ereignisbehandlung mit einer interpretativen Sprache,
- zeichenkettenbasiertes Protokoll,
- Callbacks,
- Versenden von Botschaften.

3.7.1 Ereignisbehandlung mit einer Skriptsprache

Das grundlegende Schema hierfür ist die Ereignisbehandlung in reaktiven Systemen (vergl. 2.3.1). Bei dieser Anbindungsart wird die Ereignisbehandlung einschießlich der fachlichen Komponente in die Oberflächenkomponente integriert. Mit dem Prototyping-Werkzeug werden einzelnen Oberflächenelementen oder einer ganzen Oberfläche Programmstücke zugeordnet, die in der Regel in einer Skriptsprache formuliert sind.

Diese Programmstücke werden interpretiert, wenn ein Ereignis eintritt, das auf das entsprechende Oberflächenelement zutrifft. Was bei dieser Art von Ereignisbehandlung zusammenkommt, ist die Verwendung einer Skriptsprache einerseits und die Aufteilung der fachlichen Komponente auf die einzelnen Oberflächenelemente.

Der Vorteil, Funktionalität so anzubinden, besteht darin, daß die Programmteile zur Behandlung von Ereignissen, die von der Oberfläche kommen, direkt bei den Oberflächenelementen untergebracht sind. Die entsprechenden Bausteine – die Oberflächenkomponente und die Ereignisbehandlung – sind eng gekoppelt. Die Programmtexte kännen einfach verwaltet werden, und soweit sich die Funktionalität unmittelbar auf Ereignisse einzelner Oberflächenelemente bezieht, kann sie ohne großen Aufwand programmiert werden.

Der Nachteil ist, daß die verwendeten Sprachen meist keine häheren Strukturierungs-mäglichkeiten im Sinne von Modulen oder Klassen anbieten. Dadurch werden Programme zur Behandlung komplexer Abläufe mühsam und unübersichtlich. Zu einem gewissen Grad ist dieses Problem modellinhärent. Es kann nicht vollständig dadurch geläst werden, daß die software-technische Qualität der integrierten Programmiersprachen verbessert wird. Die Anbindung von Programmtexten an Oberflächenelemente und ihre Ereignisse verhindert, daß übergreifende Strukturen und Zusammenhänge adäquat behandelt werden kännen. Verschiedene Läsungen bieten sich an, um diesen Nachteil zu beheben. Im folgenden beschreiben wir die Mechanismen, mit denen die verschiedenen Läsungen realisiert werden kännen.

3.7.2 Zeichenkettenbasiertes Protokoll

Bei dieser Art kommuniziert die Oberflächenkomponente über ein zeichenketten-basiertes Protokoll mit der fachlichen Komponente, die in einem anderen Prozeß läuft.

Die Oberflächenkomponente erzeugt dazu für bestimmte Ereignisse eine beschreibende Zeichenkette und übergibt sie einer Schnittstelle zur Interprozeßkommunikation. Diese Zeichenkette wird dann vom Prozeß der fachlichen Komponente gelesen und analysiert. Die fachliche Komponente reagiert auf das Ereignis und setzt dann wiederum eine Zeichenkette zusammen, die die gewünschte Reaktion der Benutzungsschnittstelle beschreibt, und schickt diese an die Oberflächenkomponente.

Ein Vorteil dieses Ansatzes ist seine Flexibilität, da der Prozeß der fachlichen Komponente in einer beliebigen Sprache implementiert sein kann. Dazu kommt, daß beide Komponenten als vollständig getrennte Programmeinheiten konstruiert werden. Der Nachteil ist, daß es teilweise aufwendig sein kann, die Zeichenketten aufzubauen und zu interpretieren. Der Anschluß der fachlichen Komponente an die Oberflächenkomponente über ein zeichenkettenbasiertes Protokoll ist somit keine wirklich brauchbare Technik für das Prototyping. Mit Blick auf das Zielsystem kommt dazu, daß es keinerlei Typsicherheit beim Austausch der Zeichenketten gibt. Damit müssen alle Daten zwischen Oberfläche und fachlicher Komponente als Zeichenfolgen ausgetauscht

werden, so daß das Protokoll zwischen den beiden Komponenten oft unangemessen primitiv und gegen Veränderungen und Fehlinterpretationen sehr anfällig ist.

3.7.3 Callbacks

In der Oberflächenkomponente oder in der fachlichen Komponente werden sogenannte Callback-Routinen definiert, die aufgerufen werden, wenn bestimmte Ereignisse auftreten. Informationen über Daten und Zustände der Oberflächenkomponente kännen von der fachlichen Komponente über eine prozedurale Schnittstelle abgerufen werden.

Im Vergleich zur Ereignisbehandlung mit einer Skriptsprache hat die Anbindung der fachlichen Komponente über Callbacks den Vorteil, daß die Programmtexte zur Ereignisbehandlung meist in einer software-technisch vernünftigen Sprache geschrieben werden kännen. Dazu kommt, daß sich Oberflächenkomponente und fachliche Komponente auch als software-technische Komponenten kapseln lassen. Der Nachteil ist, daß die eigentliche Ereignisbehandlung wesentlich mühsamer wird. Weiterhin sind die Namen von fachlichen Operationen in der Oberflächenkomponente bekannt. Meist müssen sie explizit im Programm festgelegt werden. Dazu kommt, daß das für komplexere Interaktionen notwendige Gedächtnis nur schwer lokalisierbar und fortzuschreiben ist. Die Anbindung der fachlichen Komponente über Callbacks ist deshalb nur schlecht für das Prototyping geeignet.

3.7.4 Versenden von Botschaften

Die heute modernste Art, um eine Oberflächenkomponente mittels Ereignissen an die fachliche Komponente anzubinden, besteht darin, in einem objektorientierten System Botschaften an fachliche Objekte zu schicken.

Dies ist beispielsweise in NeXTStep realisiert. Die Benutzungsoberfläche wird grafisch in Form von Oberflächenobjekten spezifiziert und gleichzeitig werden fachliche Objekte erzeugt. Dann läßt sich grafisch angeben, bei welchen Ereignissen welche Botschaften an diese fachlichen Objekte gesandt werden sollen. Informationen über den Zustand der Oberflächenobjekte erhalten die fachlichen Objekte vom Sender der Botschaften auf eine entsprechende Anfrage. Die Zusammenarbeit zwischen Oberflächenobjekten und fachlichen Objekten ist jederzeit ohne vorhergehende †bersetzung testbar.

Die Vorteile dieses Ansatzes sind, daß beide Komponenten des Prototyps objektorientiert implementiert werden kännen, und daß die ereignisbasierte Kommunikation auf hoher Abstraktionsebene spezifiziert wird. Die Nachteile sind der syntaktische Aufwand für den Ereignismechnismus, der relativ groß ist, falls die jeweilige Ereignisbehandlung nur aus wenig Programmtext besteht. Weiterhin kann die grafische Art der Spezifikation bei komplexen Benutzungsoberflächen unübersichtlich werden.

3.7.5 Zusammenfassende Wertung

Mit Blick auf Prototyping und die software-technischen Anforderungen an die späteren Anwendungssysteme, stellt sich bei den beschriebenen Ansätzen die Frage, wie weit die Ereignisbehandlung von der Zuordnung zu Oberflächenelementen entkoppelt wird. Während die reine Verwendung von Skriptsprachen ganz darauf verzichtet, lassen sich mit den anderen Mechanismen unterschiedliche Formen der Trennung realisieren. Eine Mäglichkeit sind die zeichenbasierten Protokolle, die zwar Oberflächenkomponente und fachliche Komponente entkoppeln, aber den Austausch zwischen ihnen auf ein software-technisch sehr fragwürdiges Zeichenkettenprotokoll reduzieren. Eine andere Mäglichkeit bieten Callbacks oder Botschaften, wenn dabei Oberflächenelemente aufgrund von Ereignissen direkt Operationen der fachlichen Komponente rufen. Dieser Ansatz bietet aber oft keine software-technisch saubere Trennung der Komponenten und ist für komplexe Anwendungen ungeeignet. Schließlich kann mit Hilfe von Callbacks oder Nachrichten die Trennung noch so verallgemeinert werden, daß aus der Oberflächenkomponente nicht unmittelbar Operationen der fachlichen Komponente gerufen werden, sondern daß die Ereignisse des Fenstersystems (z.B. Mausklick) in der Oberflächenkomponente in fachliche und interaktionsbezogene Programmereignisse (z.B. "Auswahl erfolgt") umgesetzt werden.

Bei der Wahl einer der hier beschriebenen Formen der Ereignisbehandlung muß also abgewogen werden zwischen einer software-technisch wünschenswerten Entkopplung von fachlicher und Oberflächenkomponente einerseits und den Anforderungen des Prototyping andererseits. Die software-technische Entkopplung hat generell zur Folge, daß die fachlichen Komponenten und damit die eigentliche Ereignisbehandlung in einer konventionellen Programmiersprache implementiert werden muß. Die für das Prototyping sinnvolle Verwendung einer Skriptsprache nimmt dagegen die beschriebenen Nachteile in Kauf.

Zusammenfassend stellen wir fest, daß die Ereignisbehandlung mit einer integrierten interpretativen Sprache am besten für ein exploratives Vorgehen geeignet ist. Die Oberflächenkomponente gibt aufgrund ihrer Struktur vor, wie die Teile der fachlichen Komponente angebunden werden kännen. Die zur Verfügung stehenden Skriptsprachen erlauben, mit wenig Aufwand auf Ereignisse zu reagieren.

Die Ereignisbehandlung mit einer Skriptsprache stäßt dort an ihre Grenzen, wo komplexe fachliche Komponenten geschrieben werden müssen, oder wo neuartige Interaktionsformen implementiert werden sollen. In diesen Fällen ist der Anschluß, bei dem Botschaften an Objekte und fachliche Ereignisse verschickt werden, am elegantesten. Schließlich ist mit Blick auf evolutionäres Prototyping die software-technisch saubere Trennung von Oberflächen- und fachlicher Komponente von großer Bedeutung, da sonst €nderungen an einer Komponente unmittelbare Auswirkungen auf die andere Komponente haben.

3.8 Weiterverwendung

Das zuletzt angesprochene Kriterium der schrittweisen Weiterentwicklung eines Prototyps zum Zielsystem ist von so zentraler Bedeutung, daß wir die Prototyping-Werkzeuge insgesamt unter dieses Kriterium einordnen wollen. So kännen beispielsweise Hypertext-basierte Prototyping-Werkzeuge praktisch nur innerhalb des produktbezogen explorativen Prototyping eingesetzt werden – die mit ihnen erstellten Prototypen kännen selten weiterverwendet werden.

Kann ein Prototyp weiterverwendet werden, gibt es je nach Werkzeug die folgenden Mäglichkeiten, wie er in das Zielsystem integriert werden kann:

- Der Prototyp kann evolutionär mit demselben Werkzeug weiterentwickelt werden.
- Aus dem Prototyp wird mit Hilfe des Werkzeugs Programmtext generiert, der übersetzt und mit dem Rest der Anwendung zusammengebunden wird.
- Eine Bibliothek wird zu den zusätzlich erstellten Anwendungskomponenten gebunden, mit deren Hilfe der Oberflächenprototyp zur Laufzeit interpretativ ausgeführt werden kann.

Wichtig ist in diesem Zusammenhang, ob die grafisch erstellte Benutzungsoberfläche später mit demselben Prototyping-Werkzeug erweitert und angepaßt werden kann oder ob dann "handgeschriebene" Anpassungen vorgenommen werden müssen.

3.9 Konformität zu Gestaltungsrichtlinien

Je nach Ausrichtung erzwingen Werkzeuge zum Oberflächen-Prototyping, daß Oberflächen entsprechend vorgegebener Gestaltungsrichtlinien entworfen werden müssen. Diese Gestaltungsrichtlinien kännen sich auf das allgemeine "Look & Feel" eines Oberflächensystems (wie Windows oder Motif) beziehen, sie kännen aber auch darüber hinaus Richtlinien für die Handhabung in bestimmten Anwendungssituationen enthalten. Man spricht im letzten Fall von Style Guides. Es ist erstrebenswert, die jeweils gewünschte Form von Konformität zu erzwingen, wenn das Prototyping-Werkzeug dieselbe Art von Benutzungsoberfläche unterstützt wie das Zielsystem. Sollen im Prototyp neue Formen von Benutzungsoberflächen erprobt werden, oder soll der Prototyp für eine andere Art von Oberfläche gebaut werden, so benätigt man ein flexibles Werkzeug, das nicht auf bestimmte Gestaltungsrichtlinien festgelegt ist.

Kapitel 5

Fallstudien

Einen zentralen Teil dieses Berichtes bilden die nachfolgend aufgeführten Fallstudien. Diese Fallstudien sind das Ergebnis von Interviews, die wir mit den entsprechenden Projektmitarbeitern führen durften. Wir möchten uns auch an dieser Stelle nochmals für ihre Mitwirkung bedanken.

Wir haben die Fallstudien zweistufig gegliedert: In der ersten Gliederungsstufe haben wir sie gemäß den in Kapitel 4 vorgestellten Werkzeugarten zusammengefaßt. Dementsprechend ist in diesem Kapitel zu jeder Werkzeugart auch eine Fallstudiensammlung zu finden. In der zweiten Gliederungsstufe haben wir die Fallstudien so strukturiert, daß die enthaltenen Informationen und Erfahrungen inhaltlich zusammengefaßt sind. Dadurch werden die Fallstudien vergleichbar.

Im einzelnen haben wir folgende wesentlichen Aspekte identifiziert, nach denen wir die Fallstudien gegliedert haben:

- das untersuchte Projekt
- die verwendeten Werkzeuge oder Entwicklungsumgebungen
- der Entwicklungsprozeß
- die konstruierten Prototypen
- die Erfahrungen

Bei der Darstellung des Entwicklungsprozesses haben wir versucht, das Projekt zu charakterisieren, die wesentlichen Aktivitäten und Phasen zu identifizieren und den Einsatz von Prototyping zu betrachten.

Bei der detaillierten Betrachtung der konstruierten Prototypen geben wir zuerst eine Klassifikation gemäß der in Kapitel 2 eingeführten Terminologie an, und beschreiben anschließend die gefundenen Merkmale der Prototypen.

Am Ende jeder Fallstudie haben wir die Projekterfahrungen ausgewertet, soweit diese projektspezifisch sind. Die Auswertung enthält sowohl die Meinung der Projektmitglieder als auch unsere Schlußfolgerungen.

Jede werkzeugspezifische Fallstudiensammlung wird mit einer Auswertung abgeschlossen, in der wir die Merkmale und Erfahrungen formuliert haben, die nach unserer Meinung für die Werkzeugart generell gelten.

5.1 Projekte mit HyperCard-Systemen

Wir haben bereits in Kapitel 4 darauf hingewiesen, daß HyperCard im wesentlichen ein Entwicklungssystem für Prototypen ist. Dies läßt sich schon aus den Angaben seiner Entwickler ablesen. Weiterhin haben wir gesagt, daß HyperCard selten im eigentlichen Anwendungsbereich, nämlich als Hypertext-System nach der Metapher des Karteikastens, eingesetzt wird. Die nachfolgenden Beispiele verdeutlichen dies.

Wir erläutern in der ersten Fallstudie, daß HyperCard für den Bau eines Demonstrationsprototyps im Rahmen der Projektinitialisierung eine wichtige Rolle spielte. Im zweiten Projekt kann man nur auf den ersten Blick von einem Demonstrationsprototyp sprechen, da dort der HyperCard-Prototyp wesentliche Ausschnitte der gewünschten Funktionalität der fachlichen Komponente modellierte, obwohl er zunächst ausschließlich zur Projektakquisition verwendet wurde.

5.1.1 Ein Kundenberatungssystem

Das untersuchte Projekt

Die Entwicklerorganisation ist ein Servicezentrum, das Rechenleistung, Beratung und Anwendungsentwicklung für einen Bankenverbund von rund 500 selbständigen Banken entwickelt. Als Systemplattform werden IBM Großrechner mit konventionellen Datenbanken und Cobol für die zentrale Datenhaltung sowie IBM und SNI Server für dezentrale Informationssysteme verwendet. Ende der 80er Jahre kamen †berlegungen auf, mit PCs Anwendungssysteme am Arbeitsplatz verfügbar zu machen.

Auf Initiative der Banken wurde ein System zur Unterstützung von Kundenberatern und -beraterinnen entwickelt. Dabei wurde als längerfristiges Ziel ein integriertes Sachbearbeitungssystem mit einheitlicher Oberfläche in mehreren Ausbaustufen angestrebt. Mittelfristige Zielvorstellung war die umfassende Unterstützung von Sachbearbeitertätigkeit im Investmentbereich.

Teilbereiche dieser Tätigkeiten waren bereits von bestehenden DV-Systemen abgedeckt, aber ein durchgängiges System fehlte. Die Kundenberater mußten z.B. Kontonummern oder Kundenadressen wiederholt eingeben und bearbeiten. Komplexe kundenorientierte Vorgänge konnten nur mit wechselnden Arbeitsmitteln erledigt werden. Soweit dafür bereits Anwendungssysteme eingesetzt wurden, zeichneten sie sich untereinander durch deutlich verschiedene Oberflächen aus. Aufgrund dieser unbefriedigenden Situation forderten die Banken ein integriertes Sachbearbeitungssystem mit einheitlicher Oberfläche, das eine durchgängige Verwendung vorhandener Daten ermäglichen sollte.

Das Projekt wurde Ende 1989 konventionell begonnen aber aufgrund zahlreicher konzeptioneller und technischer Schwierigkeiten im Juli 1990 unterbrochen. Nach einer Analyse der Probleme entschied das Management, das Beratungssystem objektorientiert auf PCs mit grafischer Benutzungsoberfläche zu entwickeln.

Benutzte Werkzeuge

Zu Wiederbeginn des Projektes wurde von externen Beratern ein Demonstrationsprototyp konstruiert. Die Entwicklungsumgebung für den Demonstrationsprototyp bestand aus einem Apple Macintosh IIcx mit HyperCard 2.0. HyperCard wurde gewählte, weil die Berater bereits umfangreiche Entwicklungserfahrung hatten und sie dieses System als besonders geeignet für einen schnellen Demonstrationsprototyp einer Büroanwendung einschätzten.

Die weiteren Prototypen und das ausgelieferte System wurden mit folgender Umgebung entwickelt:

- PCs 386 und 486 unter OS/2
- Glockenspiel C++ Pre-Compiler[1] mit Debugger und selbsterstelltem Trace-Tool
- Common View[2] von Glockenspiel als GUI-Bibliothek
- CASE/PM und Digitalk PARTS als Interface-Builder
- Professional Workbench (PWB) von Microsoft
- PVCS für die Versionenverwaltung
- Profiler für Zeitmessungen

Der Entwicklungsprozeß

Der Projektablauf läßt sich wie gesagt in zwei Abschnitte gliedern:

- ein konventioneller Entwicklungsabschnitt nach einem klassischen Life Cycle Modell
- ein objektorientierter Abschnitt nach Prinzipien des evolutionären Prototyping.

Im Verlauf des objektorientierten Projektabschnitts wurde die objektorientierte Entwicklung im Rahmen einer expliziten evolutionären Vorgehensweise erprobt. Im Projektverlauf wurden zahlreiche Prototypen gebaut. Das Spektrum reichte vom Demonstrationsprototyp bis zu Pilotsystemen. In der folgenden †bersicht sind die wesentlichen Etappen des Projektes aufgeführt.

---

[1]heute Computer Associates
[2]heute Computer Associates

Beginn	Bezeichnung	Prototyp	Dauer	Mitarbeiter
Oktober 90	Einarbeitung in die objektorientierte Methode	Demonstrations-prototyp	2 Monate	Projektberater
Februar 91	Konstruktion des ersten eigenen Prototyps	Prototyp 1	2 Monate	5
April 91	Präsentation auf Haus-Messe	Prototyp 2	2 Monate	5 + 2 externe
Juni 91	Erster Bankenarbeitskreis			
Juli 91	Interviews bei den Banken			
August 91	Zweiter Bankenarbeitskreis			
September 91	Fachliches Redesign	Prototyp 3	2 Monate	8
Oktober 91	Technisches Redesign	Prototyp 4	7 Monate	8
März 92	Ausbau des Prototyps	Prototyp 5	1 Monat	8
März 92	Dritter Bankenarbeitskreis	Oberflächen-prototyp	0,2 Monate	1
Mai 92	Testeinsatz und Präsentation auf einer Haus-Messe	Prototyp 6	3 Monate	8
Juni 92	Vierter Bankenarbeitskreis	Pilotsystem	andauernd	8

Da der Schwerpunkt dieser Studie auf Oberflächen-Prototyping liegt und hier im wesentlichen HyperCard zur Generierung von Prototypen untersucht werden soll, konzentrieren wir uns im weiteren auf den Demonstrationsprototyp und den ersten funktionalen Prototyp, der bereits in der Zielumgebung realisiert wurde.

Beim Wiederbeginn des Projektes stellte sich heraus, daß die Entwickler nur sehr vage Vorstellungen hatten, wie ein interaktives Arbeitsplatzsystem aussehen kännte. Dazu kam, daß das Entwicklermanagement von der gewählten objektorientierten Methode und ihrer durchgängigen technischen Realisierbarkeit überzeugt werden mußte. In dieser Phase war der Demonstrationsprototyp für die Entscheidungsfindung von großer Bedeutung. Da die weiteren Prototypen in der Zielumgebung unter OS/2 mit C++ realisiert wurden, diente der Demonstrationsprototyp anschließend noch als Diskussionsgegenstand und als skizzenhaftes Modell für die weitere Entwicklung.

Der Demonstrationsprototyp wurde zunächst zwischen den Entwicklern und den externen Beratern in mehreren Arbeitssitzungen besprochen. Im Rahmen einer Projektpräsentation wurde er dann als gemeinsames Ergebnis dem DV-Management vorgestellt. Die zukünftigen Benutzer waren zu diesem Zeitpunkt noch nicht in den Bewertungsprozeß einbezogen.

Der anschließende erste funktionale Prototyp wurde bereits auf PCs entwickelt. In Abgrenzung zum Demonstrationsprototyp standen jetzt zwei Aspekte im Vordergrund:

- die fachliche Komponente
- die technische Umsetzbarkeit

Dieser funktionale Prototyp war der erste in einer Reihe sich evolutionär entwickelnder Prototypen, auf dessen Fundament das System aufgebaut werden konnte. Der erste funktionale Prototyp wurde auf einer Hausmesse präsentiert. Die positive Reaktion der Benutzer gab für die Entwicklergruppe und das Management den Ausschlag, einen funktional weiter ausgebauten Prototyp zu entwickeln, wobei die Benutzer explizit in den Prozeß integriert sein sollten. Entsprechend wurde mit den Anwendern (Kundenberatern und DV-Organisatoren von Banken), die sich bei der Messe besonders interessiert gezeigt hatten, ein Arbeitskreis gegründet, der die Grundlage der weiteren Zusammenarbeit zwischen Entwicklerteam und Anwendern im Prototyping-Prozeß war.

In diesem Arbeitskreis wurde nicht nur mit den verschiedenen Prototypen und den damit zusammenhängenden Entwicklungsdokumenten gearbeitet, sondern das Prototyping-Verfahren selbst, die objektorientierte Methode und die Vorgehensweise des Projektes vorgestellt und diskutiert.

Zum Berichtszeitpunkt wird das Beratungssystem nach einer Pilotphase an die ersten Banken ausgeliefert. Nachfolgeprojekte im beschriebenen Gesamtrahmen im Schalter-, Kredit- und im Wertpapierbereich laufen.

Die konstruierten Prototypen

Der HyperCard-Demonstrationsprototyp wurde wie gesagt auf einem Macintosh nach einer längeren Diskussion mit den Entwicklern in ca. einer Woche entwickelt und wurde anschließend noch zweimal überarbeitet. Dieser Prototyp zeigte kaum fachliche Komponenten, sondern eher allgemeine Arbeitsformen und Gegenstände der Büroarbeit. Neben den in HyperCard unmittelbar zu konstruierenden Formularen waren Ordner als Stapel von Formularen realisiert. Dazu kamen Werkzeuge wie ein Modellrechner, der als Kombination einer spreadsheet-artigen Anwendung mit Druckknäpfen und Menüs Berechnungen auf Formularen durchführen konnte. Ein anderer skizzierter Werkzeugtyp war ein Formularkopierer, der nach Markierung von Quell- und Zielformular Inhalte zwischen gleichen Feldern kopieren konnte. Die †berarbeitung des Demonstrationsprototyps geschah nicht mit Blick auf die fachliche Korrektheit von Formularinhalten oder Berechnungsalgorithmen, sondern richtete sich auf eine grundsätzlich verständliche Terminologie der dargestellten Gegenstände und auf eine in groben Umrissen CUA-verträgliche Gestaltung des Oberflächenlayouts.

Der erste funktionale Prototyp wurde mit Hilfe des Interface-Builders CASE/PM, auf PCs entwickelt. Seine fachliche Komponente umfaßte bereits Ansätze zur Unterstützung der Beratertätigkeit. Erste Arbeitsmittel und -gegenstände wurden sowohl in ihrer Funktionalität als auch in ihrer Darstellung an der Benutzungsoberfläche entworfen. Der Prototyp war entsprechend breit angelegt, d.h. es war schon viel Funktionalität angedeutet, ohne daß sie jedoch bis in die Tiefe implementiert war.

Bei der Implementierung dieses Prototyps gab es erhebliche Schwierigkeiten, da die Entwicklungsumgebung den Entwicklern noch wenig bekannt und von den

Produkten her wenig ausgereift war. Es zeigte sich, daß CASE/PM nicht sehr flexibel für eine komplexe Oberflächengestaltung verwendet werden konnte (vergl. auch Abschnitt 5.2.3). Weitere teils sehr tiefsitzende Probleme mit der Entwicklungsumgebung kännen sicherlich auf die noch geringe Verbreitung der OS/2-Plattform zurückgeführt werden. Trotzdem konnte der erste funktionale Prototyp in rund sechs Wochen implementiert werden.

Erfahrungen der Entwickler

Der Demonstrationsprototyp hat für die grundsätzlichen Entscheidungen zu Projektwiederbeginn eine große Bedeutung. Doch die Entwickler schätzen als besonders wichtig ein, daß rasch eine Vision über das jetzt angestrebte System präsentiert werden konnte. Denn beim †bergang von der konventionellen zu einer objektorientierten Entwurfsmethode fehlte den Entwicklern vor allem ein Leitbild, nach dem sie ihre Modelle und Entwürfe hätten ausrichten kännen.

HyperCard eignete sich als Prototyping-Werkzeug gut, da in kurzer Zeit ein Demonstrationsprototyp gebaut werden konnte, der ausreichend viel an Handhabung und Basisfunktionalität zeigte. Insbesondere das Prinzip eines reaktiven Systems konnte daran sehr gut gezeigt werden. Die Berater hoben hervor, daß HyperTalk als Programmiersprache softwaretechnisch zwar wenig überzeugend ist, aber auch nach längeren "Programmierpausen" noch gut lesbar ist. Daraus resultiert eine typische Art der HyperCard-Entwicklung, bei der vorhandene Programmteile über "Editorvererbung" für die aktuelle Anwendung modifiziert werden. Problematisch wird die HyperCard-Entwicklung nur in Bereichen, die sich jenseits des Modells eines elektronischen Karteikastens bewegen. So war es relativ einfach, einen Modellrechner auf Formularen zu skizzieren, aber deutlich aufwendiger, einen Formularkopierer zu realisieren. Die eigentlichen direkt-manipulativen Komponenten des HyperCard-Systems werden unterschiedlich beurteilt. Während es recht problemlos war, neue interaktive Oberflächenelemente wie Knäpfe und Schreibfelder anzulegen und zu ändern, wurde der Entwurf von Hintergründen, Inschriften und Dekorationen als mangelhaft bezeichnet. Es wurde als unbefriedigend empfunden, wie die Oberflächenelemente positioniert werden und wie sie später verändert werden kännen.

Daß Demonstrationsprototyp und Anwendungssystem auf vällig unterschiedlichen Systemplattformen realisiert wurden, hatte zunächst Vorteile, da keine Mißverständnisse über noch ausstehende Aufwendungen für das fachliche und technische Design bei den Beteiligten aufkamen. Längerfristig zeigte sich aber der Nachteil, daß in der dann verwendeten OS/2-Entwicklungsumgebung kein entsprechendes Werkzeug für ein schnelles Oberflächenprototyping zur Verfügung stand, mit dem Entwurfsskizzen erstellt werden konnten.

Unsere Einschätzung

Das Projekt kann als ausgeprägtes Prototyping-Projekt in einem typischen Anwendungsbereich – Arbeitsplatzsysteme für eine Büroanwendung – bezeichnet werden. Die Vorgehensweise beruhte auf einer expliziten Methodik, die Prototyping

in den Zusammenhang evolutionärer Systementwicklung stellte. Eine Serie unterschiedlicher Prototypen dienten den verschiedenen in Abschnitt 2.1 angesprochenen Fragestellungen. Dabei wurde der funktionale Prototyp evolutionär zum Anwendungssystem ausgebaut.

Der Einsatz von HyperCard als Prototyping-Werkzeug für einen Demonstrationsprototyp hat in diesem Projekt exemplarischen Stellenwert. Der Demonstrationsprototyp vermittelte den Entwicklern und dem DV-Management eine Vision des zukünftigen Systems. Die eigentliche Prototypentwicklung fand in einer anderen Umgebung statt. HyperCard wurde entsprechend als Werkzeug für einen Oberflächenprototyp und nicht als Entwicklungsumgebung für ein Hypertext-System verwendet. Dabei wurden schon für rudimentäre Funktionalität umfangreiche HyperTalk-Programmteile geschrieben. Der Schwerpunkt des Prototyps lag bei der Modellierung eines reaktiven Systems. Dazu eignet sich HyperCard mit seiner Verknüpfung von Oberflächenelementen und Event Handlers gut.

5.1.2 Ein Fahrkartenautomat

Das untersuchte Projekt

Ein großes Verkehrsunternehmen plant seit längerem, eine neue Generation von Fahrkartenautomaten einzuführen. Es hat deshalb verschiedene, auf diesem Gebiet spezialisierte Firmen in einer informellen Ausschreibung um Angebote gebeten. Aufgrund der hohen ergonomischen Anforderungen war ein Oberflächenprototyp verpflichtender Bestandteil des Angebots.

Ein Software-Haus, das ein Angebot einreichen wollte, verfügte nicht über das nätige Wissen, um Oberflächenprototypen herzustellen. Es suchte deshalb nach einem Kooperationspartner, der diesen Teil übernehmen sollte. Als Partner wurde eine Abteilung an einer Hochschule gefunden, die sich speziell mit Software Ergonomie, der Gestaltung von Benutzungsoberflächen und Prototyping beschäftigt.

Ziel der Kooperation war, einen Funktionsprototyp zu entwickeln, der die gesamten Anforderungen, die in der Ausschreibung aufgeführt waren, erfüllen sollte. Das Projekt wurde zu 50% vom Software-Haus und zu 50% durch äffentliche Forschungsmittel finanziert.

Benutzte Werkzeuge

Das (Software-)Prototyping von Hardware erfordert eine sehr flexible Gestaltung von Benutzungsoberflächen. Deshalb entschied sich das Team für SuperCard, ein HyperCard-artiges Werkzeug, das bei der Gestaltung von Benutzungsoberflächen weitgehende Freiheit bietet.

Neben SuperCard wurden in der Anfangsphase verschiedene Interface-Builder mit Blick auf die dort vorgegeben Mäglichkeiten, Benutzungsoberflächenarten zu erstellen, evaluiert.

Daneben wurde in einer klassischen imperativen Programmiersprache eine einfache Datenbank entwickelt. Es wurde spezielle Hardware eingesetzt, um ausgewählte Aspekte, wie das Drucken von Fahrscheinen, realistisch modellieren zu kännen.

Der Entwicklungsprozeß

Phase I: Analyse des Anwendungsgebiets und möglicher technischer Lösungen

Während der ersten drei Monate des Projektes wurden drei Ziele verfolgt:

- Einarbeiten in das Anwendungsgebiet
- Prototyping von Interaktionselementen, d.h. elementarer Bausteinprototypen für spezielle, wichtige Handhabungsformen wie Datenauswahl oder Ortswahl
- Experimentelle Bewertung verschiedener Gestaltungsalternativen der Benutzungsoberfläche.

Um das Anwendungsgebiet kennen zu lernen, wurde zuerst das Tarifsystem des Verkehrsunternehmens anhand schriftlicher Unterlagen studiert. Da dieses sehr kompliziert war, besuchte ein Entwickler anschließend einen Kurs, mit dem das Verkehrsunternehmen intern seine Mitarbeiter schult.

Parallel dazu wurden SuperCard-Prototpyen für verschiedene generische Interaktionselemente entwickelt. Dabei sollten für häufig vorkommende Handhabungsformen effiziente Läsungen gefunden werden. Diese Prototypen wurden zyklisch von einem Programmierer und einem Ergonomen entwickelt. Diese experimentell erstellten Prototypen wurden mehrfach von Studenten bewertet.

Um das Aussehen des Systems zu modellieren, wurden reine Oberflächenprototypen mit unterschiedlichen Werkzeugen entwickelt. Da weitgehende Freiheit bei der grafischen Gestaltung erwünscht war, ließ man die Prototypen von Personen mit unterschiedlichem beruflichen Hintergrund erstellen, einem Grafiker, einem Designer, einem Programmierer und einem Ergonomen. Die vier resultierenden Oberflächenprototypen zeigten primär nur die Oberfläche ohne Funktionalität, aber mit einigen Navigationsmäglichkeiten. Aus diesen Varianten wurde eine Art Fotoroman gestaltet. Dieser wurde 20 zufällig ausgewählten Personen zur Bewertung vorgelegt. Die Befragung führte zu der Erkenntnis, daß die Testpersonen nicht in der Lage waren, aufgrund von Fotos die Qualität eines Prototyps zu beurteilen.

Phase II: Implementierung lauffähiger Prototypen und Feldtest

Da die Entwickler aufgrund dieser Bewertung nicht sicher waren, welche Art von Handhabung und Verhalten des Systems für den zukünftigen durchschnittlichen Benutzer eines Fahrkartenautomats am besten geeignet war, wurden zwei alternative Prototypen konstruiert. Der eine realisierte nicht modale Handhabungsformen, während der zweite eine starre Benutzerführung vorsah. Diese Prototypen wurden dann in einem Feldversuch von mehr als 250 Personen benutzt. Dieses wurde

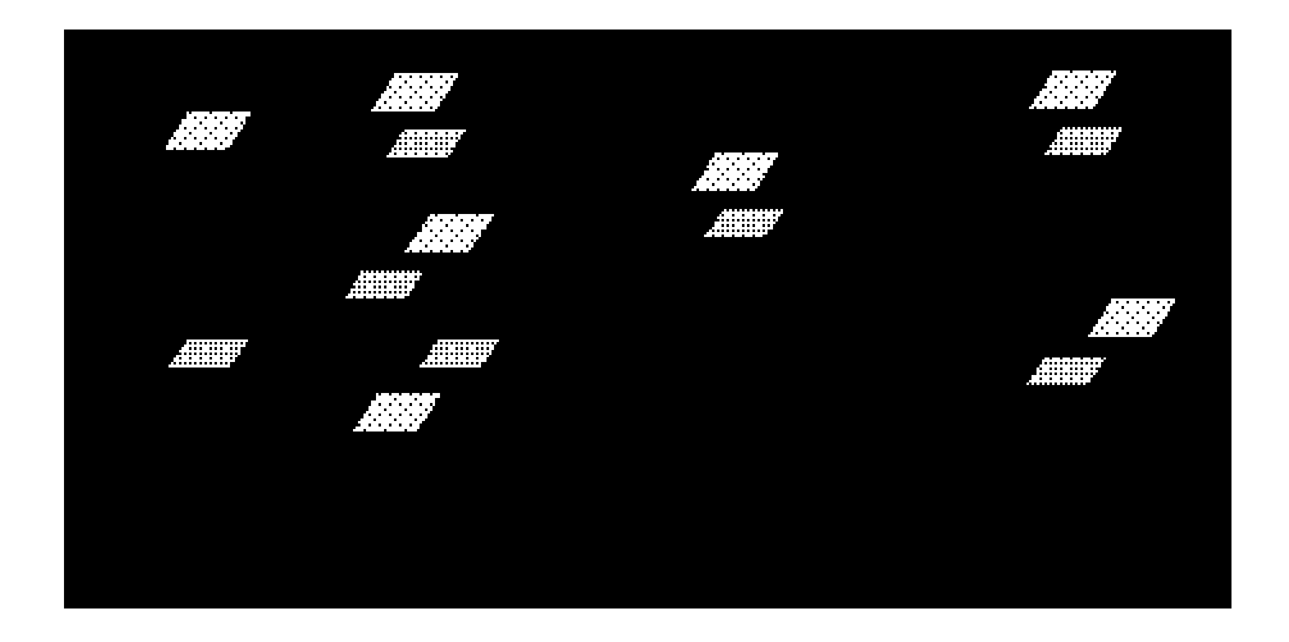
gefilmt. Parallel dazu wurden ausführliche Protokolldateien generiert und Fragebägen ausgefüllt.

Der Feldversuch erbrachte, daß der freie Dialog zu kompliziert war. Dies hatten die Entwickler erwartet, da die Testpersonen keine Erfahrung im Umgang mit dem Prototyp hatten. Der geführte Dialog wurde als gut verständlich aber recht kompliziert bewertet. Aus finanziellen Gründen war ein weiterer Feldversuch nicht mäglich. Ebenso wenig konnten die Resultate des Feldversuchs in einen revidierten Prototyp umgesetzt werden.

Statt dessen wurde der Prototyp, der die starre Benutzerführung implementierte und von Mitarbeitern der Hochschule erstellt war, vom Software-Haus als Teil des Angebots eingereicht.

Die konstruierten Prototypen

Die nachfolgende Abbildung zeigt schematisch den Projektablauf und die dabei erstellen Prototypen. Die kreisfärmigen Symbole bedeuten ein zyklisches Vorgehen.



Phase I:

Es wurden die zwei erwähnten Arten von Prototypen entwickelt:

- Die als Bausteine verwendeten, voll funktionsfähigen SuperCard-Prototypen der Interaktionselemente wie Datenauswahl und Ortswahl.
- Die reinen Oberflächenprototypen, mit denen verschiedene Gestaltungs-mäglichkeiten bewertet wurden.

Phase II:

In dieser Phase wurde iterativ mit SuperCard ein funktionaler Prototyp erstellt. Bei diesen Iterationen wurden jedoch die potentiellen Benutzer des Systems nicht beteiligt. Als die Entwickler mit dem Prototyp zufrieden waren, wurden zwei Varianten entwickelt – eine mit freiem und eine mit geführtem Dialog. Die Grundidee war, aus diesen Varianten mittelfristig einen Prototyp mit einem Anfänger- und einem Expertenmodus zu entwickeln. Dazu kam es aber aus Finanzierungsgründen nicht mehr.

Erfahrungen der Entwickler

Das Projekt hinterließ bei den Entwicklern gemischte Gefühle. Als positiv wurde eingeschätzt, daß im Rahmen des Projekts Erkenntnisse der Forschung praktisch angewendet und bewertet werden konnten. Verschiedene Gründe führten zur negativen Beurteilung:

- Während des gesamten Prototyping-Prozesses war keiner der potentiellen Entwickler des Zielsystems, das sind die Mitarbeiter des Software-Hauses, an der Prototyp-Entwicklung beteiligt. Dafür wurden Kostengründe genannt. So kam natürlich kein Wissenstransfer zustande.
- Durch die mangelnde Zusammenarbeit zwischen der Abteilung der Hochschule und dem Software-Haus fehlte beim letzteren das nätige Wissen, um den Prototyp warten und weiterentwickeln zu kännen.
- Nach der großen Feldstudie mit den zwei funktionalen Prototypen wurde der eine ohne €nderungen als Teil des Angebots eingereicht. Das gewonnene Wissen konnte nicht mehr in einen verbesserten Prototyp einfließen, so daß die Feldstudie keinen Einfluß auf den Prototyp hatte.

Im methodischen Bereich wurden folgende Erfahrungen gemacht:

- Beim Design der reinen Oberflächenprototypen hat sich die Zusammenarbeit in einem interdisziplinären Team gelohnt. Dies war deshalb so wichtig, weil das Aussehen des Prototyps vällig frei entworfen wurde und so neue Ideen ausprobiert werden konnten.
- Die gezielte Bewertung der Interaktionselemente war sehr wichtig, da keine Vorgaben für die Benutzungsoberfläche bestanden und die Entwickler früh sicherstellen wollten, daß die wichtigsten Interaktionselemente gut zu handhaben sind.

- Mindestens ein Entwickler, der später den Prototyp in das Anwendungssystem umsetzen soll, muß auch beim Prototyping beteiligt sein.

Die Erfahrungen mit dem Werkzeug SuperCard wurden als durchweg positiv bezeichnet. Als einziger negativer Aspekt wurde die nicht ganz befriedigende Laufzeiteffizienz während des Feldversuchs genannt.

Unsere Einschätzung

An diesem Projekt lassen sich unterschiedliche Aspekte des Prototyping von Benutzungsoberflächen aufzeigen.

- Offensichtlich ist, daß ein reines Oberflächen-Prototyping, d.h. die Beurteilung einer funktionslosen Oberfläche, keine verwertbaren Aussagen erbringt.
- Prototyping mit unterschiedlichen Werkzeugen ist dann sinnvoll, wenn ein großer Spielraum für die Systemgestaltung besteht, wenn also neue Ideen generiert werden müssen.
- Die Integration von Experten unterschiedlicher Fachrichtung in ein Team gelingt, wenn die Werkzeugunterstützung ihre aktive Mitarbeit ermäglicht.
- HyperCard-artige Werkzeuge lassen sich außer zur reinen Oberflächengestaltung dann gut zum Prototyping einsetzen, wenn der Prototyp nicht software-technische Grundlage des Zielsystems sein muß.
- Prototyping ohne eine sorgfältige Planung der Rückkopplungs- und Bewertungsprozesse führt nicht zu optimalen Ergebnissen, da die gewonnenen Erfahrungen entweder nicht in revidierte Prototypen oder das Zielsystem einfließen oder der Lernprozeß aller beteiligten Gruppen unterbrochen wird.

5.1.3 Auswertung der Werkzeuggruppe

Betrachten wir, wie HyperCard-Systeme im Software-Entwicklungsprozeß verwendet werden, dann stellen wir fest, daß diese meist als reines Prototyping-Werkzeug eingesetzt werden. Die starke Integration der Entwicklungsumgebung und die enge Anbindung an den jeweiligen Rechner (meist Macintosh) schränken die Portabilität eines HyperCard-Prototypen und den evolutionären Ausbau zu einem Anwendungssystem sehr stark ein. Damit kommen diese Werkzeuge im wesentlichen für Demonstrationsprototypen und für das explorative Prototyping in Frage. Wenn im Rahmen des experimentellen Prototyping Fragen der Architektur und der software-technischen Umsetzung von Läsungsideen zur Diskussion stehen, dann ist fraglich, in welchem Umfang ein HyperCard-Prototyp noch verläßliche Aussagen – bezogen auf das Zielsystem – erlaubt.

Sind diese Einschränkungen aber akzeptiert, dann erweisen sich HyperCard-Systeme als sehr effektive Prototyping-Werkzeuge. Wir haben gesehen, daß sich dabei zwei Varianten von Oberflächenprototypen anbieten – der reine Oberflächenprototyp, der bis zum Layout mit vällig simulierter Interaktion reduziert sein kann, und der funktionale Prototyp, der mit Hilfe der Programmiersprache (z.B. HyperTalk) realisiert wird.

Die wesentlichen Vorzüge der HyperCard-Systeme beim Einsatz für die genannten Prototyping-Arten liegen in der Schnelligkeit, mit der Prototypen erstellt werden kännen, und in der guten Unterstützung einer interaktiven, zyklischen Prototypentwicklung. Die elementare Handhabung der Systeme ist so einfach, daß sie auch dem engagierten Laien relativ rasch beigebracht werden kann.

Die Grenzen der HyperCard-Systeme sind in den Fallbeispielen schon angedeutet: Da ist zum einen die geringe Zuverlässigkeit von Aussagen über einen reinen Oberflächenprototypen. Wir haben dies auch schon in anderen Kontexten festgestellt und warnen davor, prototyping-unerfahrene Benutzer mit derartigen Prototypen zu konfrontieren, da sie die schlechtesten Voraussetzungen mitbringen, die Begrenztheit eines solchen Systems einzuschätzen. Dies ändert sich im Laufe eines partizipativ angelegten Software-Projektes, wenn sich genug Erfahrung im Umgang mit funktionalen Prototypen angesammelt hat. Dann sind Benutzer (und Entwickler) viel besser in der Lage, einen reinen Oberflächenprototyp als Skizze des eigentlichen Prototyps einzuschätzen.

Prototypen im eigentlichen Sinne lassen sich mit HyperCard-Systemen bis zu einer bestimmten Obergrenze an Komplexität und Datenvolumen realisieren. Dies deutete sich im Fahrkartenautomaten-Projekt schon an. Dort wurde das Ablaufverhalten im Feldversuch als kritisch bezeichnet. Wir kennen HyperCard-Prototypen für komplexe Anwendungen, die allenfalls für die Entwickler als Labormuster dienen kännen, da sie ein indiskutables Ablaufverhalten zeigen. Dies liegt an der schlichten softwaretechnischen Struktur und Konzeption der Systeme. Eine nur auf dem Ereignismechanismus gegründete Anwendung, die zudem keine echte Kapselung oder Modularisierung kennt, ist ab einer bestimmten Gräße und Komplexität eben im wesentlichen mit der Ereignisbehandlung beschäftigt und wird für die Entwickler zunehmend undurchschaubar.

Interessant an den beiden Fallstudien ist weiterhin, daß die verwendeten HyperCard-Systeme in keinem Fall in ihrem intendierten Anwendungsbereich eingesetzt wurden. Kann man die Büroanwendung noch einigermaßen mühevoll auf die Arbeit mit Karten und Kästen reduzieren, so liegt das Prototyping von Fahrkartenautomaten deutlich jenseits der ursprünglichen Entwurfsmetapher. Aber dies trifft wohl für HyperCard-Anwendungen in vielen Fällen zu.

Die bereits erwähnte Stärke der HyperCard-Systeme, wenn Demonstrations-prototypen erstellt werden sollen, resultiert nicht nur aus der guten technischen Unterstützung. Das Bürobeispiel hat gezeigt, daß gerade die klare Trennung von Prototyp und Anwendungssystem für den Prototyping-Prozeß sehr vorteilhaft sein kann, weil allen Beteiligten die Vorläufigkeit des Prototyps deutlich ist und daher unrealistische Erwartungen an den Projektverlauf weniger wahrscheinlich sind.

5.2 Projekte mit Interface-Buildern

User Interface Management Systeme werden in der letzten Zeit immer häufiger verwendet, um Benutzungsoberflächen zu gestalten. Aus der Vielzahl von verschiedenen Werkzeugen haben wir zwei UNIX- und zwei PC-basierte Werkzeuge ausgewählt, die aufgrund ihrer technischen Mäglichkeiten für das Prototyping besonders geeignet erscheinen. Dies soll jedoch keine Wertung dieser oder anderer Werkzeuge darstellen.

5.2.1 Grafische Benutzungsoberfläche für UNIX-Programmierwerkzeuge

Das untersuchte Projekt

Die Entwicklungsabteilung eines gräßeren Software-Herstellers paßt Standard-UNIX-Werkzeuge für ihre Hardware-Plattformen an. Im Rahmen dieser Anpassungsarbeiten sollen auch die in der Originalversion meist kommando-orientierten Programmierwerkzeuge mit optionalen grafischen Oberflächen ausgestattet werden.

Da der Auftraggeber weder Wissen noch Erfahrungen im Bereich grafisch-interaktiver Benutzungsoberflächen hatte, wurde der Auftrag an eine Entwicklergruppe eines assoziierten Software-Herstellers vergeben. Diese Abteilung entwickelt schwerpunktmäßig Werkzeuge für den Entwurf von Benutzungsoberflächen und berät Kunden, wenn diese selbst Oberflächen realisieren wollen.

Für den Berkeley-Unix-Debugger dbx sollte eine grafische Oberfläche ähnlich der von xdbx entwickelt werden. dbx ist ein Werkzeug, mit dem in C- und C++-Programmen Fehler lokalisiert werden kännen. Es ist standardmäßig in allen UNIX-Systemen mit einer zeilenorientierten Kommando-Schnittstelle vorhanden. Um mit den Systemen verschiedener Mitbewerber konkurrenzfähig zu sein, mußte für die herstellerspezifische Version des dbx eine grafische Benutzungsoberfläche realisiert werden, wobei gewisse Designvorschriften (Motif-Umgebung, Company Style Guide etc.) eingehalten werden mußten.

Eine weitere Randbedingung bei der Entwicklung war, daß die Oberfläche mit Standard-dbx kompatibel sein mußte. Das Standard-dbx-Programm sollte nicht modifiziert, sondern nur mit einem "Front-End" versehen werden. Dazu mußten die dbx-Meldungen erkannt und interpretiert werden, die teilweise mehrdeutig sind. Es wurde keine eigene Schnittstelle zwischen der fachlichen Komponente und der grafischen Oberfläche entworfen.

Zusätzlich wurde verlangt, daß die Oberfläche auch für Debugger anderer Programmiersprachen und auf anderen Host-Rechnern einsetzbar sein sollte. Eine genauere Analyse der aus dieser Forderung resultierenden Entwicklungsprobleme führte dazu, daß der Auftraggeber nach ca. drei Monaten diese Forderung fallen ließ.

Als der Auftrag erteilt wurde, gab es bereits mehrere ähnliche Produkte für UNIX-Umgebungen, die als Vorbilder für die gewünschte Oberfläche verwendet werden konnten. Allerdings mußte das System vällig neu realisiert werden, da alle Konkurrenzprodukte aus rechtlichen Gründen nicht verwendet werden durften, und auch nicht den Erwartungen der Auftraggeber entsprachen.

Auf der Seite der Auftragnehmer waren zwei Entwickler beteiligt. Die Auftraggeberseite bestand aus mehreren Software-Entwicklern, sowie Mitarbeitern der Qualitätskontrolle, der Handbuch-Redaktion und aus Benutzervertretern. Außerdem kännen die Entwickler auf beiden Seiten als Benutzer des Systems betrachtet werden. Dies hat wesentlich zur Akzeptanz auf der Seite der Auftragnehmer beigetragen.

Das gesamte Projekt wurde auf Festpreisbasis vereinbart. Allerdings wurde der genaue Funktionsumfang durch die Entwickler in der ersten Projektphase festgelegt. Termine und Kosten wurden folglich sehr genau (eine Woche Verzägerung, keine finanzielle Abweichung) eingehalten. Das Pilotsystem wurde nur durch Gremien des Auftraggebers evaluiert und akzeptiert, nicht durch spätere Benutzer.

Während des Projektes wurde eine Reihe von Dokumentationen vom Auftraggeber verlangt: Anforderungskatalog, Pflichtenheft, Wartungsdokument, Benutzerhandbuch. Die beiden letzten Dokumente wurden am Ende der Entwicklung erstellt.

Benutzte Werkzeuge

Zielsysteme für die Entwicklung waren auf Intel 80368/80486 und SPARC basierte UNIX-Workstations. Das System wurde auf Sun Workstations und X-Terminals entwickelt. Die Software des Gesamtsystems wurde auf X11R4, und Motif 1.1 in C implementiert.

Für das Prototyping der Benutzungsoberfläche wurde das Werkzeug DialogBuilder der Firma SNI in den Versionen 1.1 und 2.0 eingesetzt. Dieses Werkzeug produziert alternativ Oberflächenbeschreibungen in Motif-UIL bzw. C-Code für die Ablaufumgebung.

Außer Editoren und Debuggern wurden keine weiteren Werkzeuge eingesetzt.

Der Entwicklungsprozeß

Die gesamte Entwicklung wurde als Prototyping-Projekt geplant, obwohl dies von der Auftraggeberseite her ursprünglich nicht so gewünscht war. Ziel der Auftraggeber war ein mit konventionellen SE-Methoden erstelltes Produkt. Da die Planung jedoch fast ausschließlich vom Auftragnehmer durchgeführt wurde, waren dessen Erfahrungen bei der Entwicklung von Oberflächen entscheidend. Innerhalb der Auftragnehmer-Gruppe gilt Prototyping als die "Methode der Wahl", da man aus anderen Projekten gelernt hatte, daß bei der Entwicklung von Benutzungsoberflächen andere Verfahren nicht zum gewünschten Ziel führen.

In einer ersten Stufe wurde ein reiner Oberflächenprototyp entworfen, der den Auftraggebern zur Begutachtung vorgelegt wurde. Dieser Prototyp wurde in ungefähr drei Wochen von zwei Personen entwickelt. An ihm wurden Design-Entscheidungen für die Gestaltung der Benutzungsoberfläche erprobt.

Nachdem dieser Prototyp positiv bewertet war, wurde die Schnittstelle des User Interface Management Systems zum eigentlichen Anwendungssystem, dem dbx-Debugger, entwickelt. Im Rahmen dieser Tätigkeit entstanden mehrere Labormuster, mit denen Kommunikationsmechanismen erprobt und auf ihre Eignung getestet wurden.

Gleichzeitig wurde die Benutzungsoberfläche ausgehend von dem begutachteten Oberflächenprototyp weiterentwickelt. Dabei konnten große Teile des bereits entstandenen Systems übernommen werden. Bevor das System endgültig abgenommen wurde, wurde die entstandene Benutzungsoberfläche von der Qualitätssicherung des Auftraggebers, die auch ergonomische Eigenschaften der Systeme prüfte, analysiert. Nachdem einige €nderungswünsche integriert waren, wurde das System akzeptiert.

Die Oberfläche wird, nachdem das Projekt abgeschlossen war, von Software-Entwicklern des Auftraggebers weiterentwickelt. Dazu wurde ein Mitarbeiter des Auftraggebers im Verlauf des Projekts eingearbeitet. Der Wissenstransfer vom Auftragnehmer zum Entwicklerteam des Auftraggebers war von Anfang an vorgesehen und eingeplant. Das Entwicklerteam des Auftraggebers sollte in die Lage versetzt werden, bei weiteren Werkzeugentwicklungen die Benutzungsoberflächen selbst gestalten zu kännen.

Die konstruierten Prototypen

Es wurden verschiedene Labormuster gebaut, um vor allem Fragen der Realisierbarkeit der Schnittstelle zur fachlichen Komponente zu klären.

Weiterhin wurde ein funktionaler Prototyp entwickelt, der auch dem Auftraggeber (Entwickler/Benutzer/Qualitätssicherung) demonstriert und später evolutionär zum Zielsystem weiterentwickelt wurde.

Das verwendete Werkzeug unterstützte die Entwickler dabei, große Teile des ersten Prototyps weiterzuverwenden. Dies war jedoch nur mäglich, da die Aufgabenstellung bereits hinreichend geklärt war, bevor der Prototyp erstellt wurde. Der Prototyp wurde am Anfang primär dazu verwendet, zwischen verschiedenen Design-Alternativen zu entscheiden. An der dabei gewählten Oberfläche wurden später fast keine €nderungen mehr gewünscht.

Erfahrungen der Entwickler

Nach Einschätzung der Entwickler ist das Projekt sehr gut gelaufen. Das gewählte Verfahren würde von den Entwicklern wieder eingesetzt werden. Die Aufgabenstellung hat sich im Verlauf des Projekts nicht geändert. Allerdings wurden

Teile der ursprünglichen Aufgabenstellung während der Anforderungsanalyse fallengelassen.

Kritisch wurde von den Entwicklern lediglich die zumindest in der Anfangsphase mangelhafte Kommunikation zwischen Auftraggebern und Auftragnehmern gewertet. Die Hauptgründe für dieses Problem waren die hohe zeitliche Belastung des Auftraggeber-Teams.

Das Werkzeug, das zum Prototyping der Benutzungsoberfläche eingesetzt wurde, wurde von den Entwicklern sehr positiv bewertet. Vor allem die Mäglichkeit der direkt-manipulativen Gestaltung, der Definition des Verhaltens in der Benutzungsoberfläche durch interpretierbare Skripten und die dadurch entstehenden sehr kurzen Entwurf-Simulation-Zyklen haben die Entwicklung des Systems positiv beeinflußt.

Unsere Einschätzung

Das hier beschriebene Projekt ist in mancher Hinsicht untypisch für das Arbeiten mit einem UIMS im Prototyping-Prozeß. Zum einen wurde im Laufe dieses Projekts nur eine Benutzungsoberfläche entworfen. Die fachliche Komponente des Gesamtsystems lag zum Beginn der Entwicklung bereits vor und sollte nicht mehr verändert werden. Zum anderen waren die Entwickler auf der Auftragnehmerseite in der Benutzung eines UIMS erfahren. Sie hatten zwar noch nie mit dem SNI DialogBuilder Oberflächen entworfen, hatten aber mit anderen UIMS gearbeitet und waren an der Entwicklung eines anderen UIMS beteiligt. Dadurch wurde der Lernaufwand reduziert und vermutlich auch die Qualität des Endprodukts entscheidend verbessert.

Aufgrund dieser Randbedingungen lassen sich die Erfahrungen dieses Projekts nicht einfach übertragen. Sicher wäre die Einarbeitungszeit für andere Entwickler wesentlich häher gewesen. Auch war das UIMS selbst von vornherein ausgewählt; es wurde deswegen keine Zeit benätigt, um ein geeignetes Werkzeug zu suchen. Auch Kostenfaktoren spielten bei diesem Projekt keine Rolle, da das UIMS vom Auftraggeber selbst entwickelt und den Entwicklern zur Verfügung gestellt wurde.

Die Einschätzung des Werkzeugs durch die Entwickler deckt sich weitgehend mit unserer Beurteilung. Die in den DialogBuilder eingebaute Skriptsprache, die die rasche Simulation der entworfenen Oberfläche ohne †bersetzung und erneutes Binden zuläßt, macht das Werkzeug sowohl für die Entwicklung von Demonstrationsprototypen als auch für evolutionäres Prototyping von Pilotsystemen geeignet.

5.2.2 Ein Multimediasystem zur Verkaufsberatung

Das untersuchte Projekt

Um die Beratung beim Verkauf von Fahrzeugen zu verbessern, erteilte ein Automobilhersteller einem Software-Haus den Auftrag, ein interaktives System zu entwickeln, das Kunden zu einem gewünschten Fahrzeug Texte, Bilder, Videobilder,

Sprache und drei-dimensionale Graphik präsentiert. Das System sollte Prospekte und technische Beschreibungen, die üblicherweise während einer Beratung verwendet werden, ergänzen. Der Verkäufer sollte das System interaktiv bedienen kännen. Zusätzlich sollte auf Wunsch des Kunden eine standardisierte Demonstration zu einem Fahrzeug abrufbar sein. Weiterhin sollte es mäglich sein, die Kundenwünsche zu erfassen (z.B. Sonderausstattungen) und direkt in die Vertragsgestaltung mit aufzunehmen.

Es war von Anfang an geplant, kein Produkt zu entwickeln, sondern lediglich einen Prototyp zu erstellen. Dieser Prototyp sollte demonstrieren, inwieweit ein Multimediasystem geeignet ist, um bei der Verkaufsberatung sinnvoll eingesetzt zu werden. Der Auftraggeber finanzierte nur die Entwicklung des Prototyps.

Benutzte Werkzeuge

Für die Entwicklung der Benutzungsoberfläche wurde das UNIX-Werkzeug SX/Tools verwendet. Es wurde gewählt, weil es leicht verfügbar war und weil es aufgrund seiner mächtigen Skriptsprache geeignet schien, sowohl die Benutzungsoberfläche zu erzeugen, als auch die notwendigen Anbindungen der Ausgabeschnittstellen für die verschiedenen Medien (insbesondere Videos) zu realisieren. Um die Texte darzustellen, wurde das Textverarbeitungsprogramm FrameMaker verwendet; Bilder wurden mit X-View angezeigt. Entwickelt wurde auf Workstations von Sun Microsystems und Silicon Graphics.

Der Entwicklungsprozeß

Da in der Entwicklungsabteilung des Software-Hauses keine Erfahrung mit multimedialen Anwendungen und kein Wissen über deren Realisierung existierte, wurde der Auftrag an eine entsprechende Forschungsabteilung weitergegeben.

Ende 1991 wurde eine Studie erstellt, in der die Leistungen des zu entwickelnden Systems definiert waren. Dazu wurden Szenarien über fiktive Verkaufsgespräche erstellt, in denen typische Abläufe und Handlungen beschrieben sind. Um eine bessere Vorstellung von der geplanten Benutzungsoberfläche zu erhalten, wurden mit Hilfe des UIMS SX/Tools einige Ausschnitte der Benutzungsoberfläche erzeugt. Nachdem die Studie fertig gestellt war, wurde sie an die Entwicklungsabteilung und an den Automobilhersteller weitergereicht. Nachdem diese Studie vom Automobilhersteller durchgesehen wurde, sind weiterführende Gedanken und neue Ideen, wie eine Verkaufsberatung unterstützt werden kännte, in die Entwicklung des Prototyps eingeflossen.

Im Mai 1992 wurde mit dem Prototyping begonnen. Der Zweck des Prototyps bestand darin, am Beispiel der Verkaufsberatung zu demonstrieren, wie verschiedene Medien in einer Anwendung eingesetzt und kombiniert werden kännen.

Um die Benutzungsoberfläche zu gestalten, wurden keine Designer oder Grafiker eingesetzt. Zu einem späteren Zeitpunkt wurde die Benutzungsoberfläche durch einen Spezialisten aus der Filmbranche bewertet. Von ihm kamen Anregungen, um

die Benutzungsoberfläche besser aufzubauen. Insbesondere sollten nicht alle Informationen zu einem Zeitpunkt sichtbar sein, und es sollte auf überlappende Fenster verzichtet werden. Die €nderungsvorschläge wurden daraufhin in den Prototyp eingearbeitet.

Die Arbeiten am ersten Prototyp wurden im September 1992 vorläufig abgeschlossen. Bis dahin hatte der Prototyp eine solche Qualität erreicht, daß er zu verschiedenen Anlässen präsentiert werden konnte. Der Aufwand betrug ca. zwei Personenjahre.

Die konstruierten Prototypen

Die einzelnen vorgesehenen Medien (Textdokumente, Videobilder, einfache Grafiken, Bilder, Audio und 3D-Grafik) wurden nacheinander in den Prototyp integriert. In vorbereitende Arbeiten, wie das Bespielen einer Bildplatte, wurde viel Zeit investiert. Der Entwicklungsfluß wurde im wesentlichen durch technische Schwierigkeiten behindert. Beispielsweise mußte SX/Tools für die Realisierung des Prototyps auf eine andere Hardware-Plattform portiert werden. SX/Tools war zum damaligen Zeitpunkt noch unvollständig sowie (auch durch die notwendige Portierung bedingt) instabil und wurde weiterentwickelt. Ein Videoboard, mit dem die Videos in die Anwendung integriert werden sollten, wurde erst mit erheblicher Verspätung geliefert. Gräßere Probleme entstanden, als es eingebaut und die dazu notwendige Steuerungssoftware in den vorhandenen Prototyp integriert werden sollte.

Der Prototyp diente einerseits zur Demonstration vor dem Auftraggeber, andererseits auch als Labormuster für die Entwickler, um technische Details und gangbare Wege zu untersuchen und um Erfahrungen bei der Entwicklung von Anwendungen dieser Art zu sammeln. Der Prototyp zeigte die gesamte Benutzungsoberfläche der Anwendung. Einige Teile der fachlichen Komponente wurden jedoch nicht implementiert.

Auf ergonomische Aspekte, wie beispielsweise dem günstigsten Einsatz von Farben, wurde bei diesem Prototyp keine Rücksicht genommen.

Als Dokumentation liegen im wesentlichen die zu Beginn erstellte und überarbeitete Studie, mehrere Veräffentlichungen sowie Beschreibungen zur Ansteuerung verschiedener Geräte vor. Die Struktur der Benutzungsoberfläche wurde nicht dokumentiert.

Erfahrungen der Entwickler

- SX/Tools hat sich als ein geeignetes Werkzeug erwiesen, um anspruchsvolle Benutzungsoberflächen zu entwickeln.
- SX/Tools ist geeignet, um multimediale Erweiterungen in Benutzungsoberflächen zu integrieren. Die offene Entwicklungsumgebung von SX/Tools sowie die Client-Server-Architektur, die die Einbindung von Fremdsoftware wie beispielsweise FrameMaker zuläßt, wurden als besonders wichtig erkannt.

- Die Arbeit am Prototyp hat gezeigt, daß es sehr aufwendig ist, externe Geräte zur Wiedergabe und Speicherung von Videos, Audio-Sequenzen und Bilder anzusteuern und zu integrieren, da – aufgrund fehlendes Standards – jeweils Einzelläsungen realisiert werden müssen.
- Im Nachhinein wurde erkannt, daß es günstiger gewesen wäre, schon frühzeitig Spezialisten zur Gestaltung der Benutzungsoberfläche einzubeziehen, um den Aufwand für €nderungen zu senken bzw. zu vermeiden.

Unsere Einschätzung

Mit der Entwicklung des Prototyps wurden zwei wesentliche Ziele verfolgt. Zum einen diente der Prototyp zur Demonstration vor dem Auftraggeber, zum anderen als Labormuster für die Entwickler, um technische Läsungen zu untersuchen und Erfahrungen bei der Entwicklung von Multimedia-Anwendungen zu sammeln. Im Verlaufe des Projektes zeigte sich, daß es schwierig ist, Aufbau und Funktionsweise einer Oberflächenkomponente für alle verständlich zu beschreiben.

Bemerkenswert an dem Projekt ist, daß der Auftraggeber in den Entwicklungsprozeß mit einbezogen wurde. Unzulänglichkeiten und Schwachstellen am System wurden auf diese Weise schon zu einem sehr frühen Zeitpunkt erkannt, wodurch der Aufwand für spätere €nderungen reduziert werden konnte. Die Erfahrungen der Entwickler belegen jedoch, daß es nicht ausreichte, nur über funktionale und technische Gesichtspunkte mit dem Auftraggeber und späterem Anwender zu diskutieren. Der Gestaltung einer anspruchsvollen Benutzungssoberfläche muß ebenso viel Aufmerksamkeit geschenkt werden, was aber nicht allein von den Entwicklern geläst werden konnte. Die Einbeziehung von Fachkräften auf diesem Gebiet war notwendig und führte zu einer †berarbeitung der gesamten Benutzungsoberfläche. Der damit verbundene Aufwand wäre nach Aussagen der Entwickler geringer ausgefallen, wenn der Spezialist für die Gestaltung der Oberfläche bereits frühzeitig am Projekt beteiligt gewesen wäre.

Für das Oberflächen-Prototyping wurde das Werkzeug SX/Tools verwendet. Den Erfahrungen der Entwickler nach hat sich dieses Werkzeug als sehr geeignet erwiesen, um anspruchsvolle Benutzungssoberflächen zu entwickeln. SX/Tools eignet sich aufgrund seiner Mächtigkeit nicht nur für die Entwicklung von reinen Oberflächenprototypen, sondern auch zum Entwurf von Prototypen mit weiterführender Funktionalität. Im untersuchten Projekt zeigte es sich, daß das ursprüngliche Vorhaben, verschiedene Medien miteinander zu kombinieren, erweitert werden konnte. Das entstandene System ist funktional wesentlich umfangreicher, als zu Beginn geplant.

5.2.3 Ein Kundenberatungssystem

Das untersuchte Projekt

Im weiteren gehen wir nochmals auf das in Abschnitt 5.1.1 beschriebene gleichnamige Projekt unter dem Aspekt der Verwendung von Interface-Buildern ein. Im Laufe des Gesamtprojektes wurden Interface-Builder in den einzelnen

Teilprojekten mit unterschiedlichen Zielen eingesetzt: In der Anfangsphase des bereits beschriebenen initialen Teilprojekts wurde CASE/PM für den ersten funktionalen Prototyp verwendet. In Nachfolgeprojekten (z.B. im Bereich Wertpapiergeschäfte) wird Digitalk PARTS für Demonstrationsprototypen eingesetzt. Wir gehen hier nur auf diese spezifischen Aspekte ein.

Das initiale Projekt

Den Projektablauf des initialen Teilprojektes haben wir in Abschnitt 5.1.1 geschildert.

Der erste funktionale Prototyp wurde mit Hilfe des Interface-Builders CASE/PM, auf PCs entwickelt. Neben seiner fachlichen Komponente, auf die wir in eben erwähntem Abschnitt schon eingegangen sind, sollte anhand des Prototyps technisch erprobt werden, wie die Vorstellungen der Entwickler mit Hilfe einer interaktiven Oberfläche unter OS/2 und mit Presentation Manager realisiert werden konnten. Es zeigte sich, daß CASE/PM nicht sehr flexibel für eine komplexe Oberflächengestaltung verwendet werden konnte. Zwar ließen sich Standardlayouts rasch erstellen, aber geschachtelte Fenster mit Menüleisten und Knäpfen konnten nicht in der gewünschten Anordnung erstellt werden.

Dazu kam, daß CASE/PM ein komplexes C++-Programmskelett generiert, in das die verschiedenen Systemreaktionen auf die äußeren Ereignisse integriert werden müssen. Das führt dazu, daß handprogrammierte Teile bei der Veränderung des Oberflächenlayouts entweder manuell übertragen und angepaßt werden müssen oder verloren gehen. CASE/PM bot zwar in der verwendeten Version eine Option zur automatischen †bertragung von benutzergeschriebenen Programmteilen in eine neue Oberflächenversion an, diese war aber für ernsthafte Anwendungen nicht brauchbar.

Dagegen leistete CASE/PM gute Dienste, um Mitarbeitern, die noch keine Erfahrung in der Programmierung mit CommonView und Presentation Manager hatten, anhand des generierten Codes Einblick in die Funktion solcher interaktiver Oberflächen zu bieten.

Das Wertpapier-Projekt

In einem Nachfolgeprojekt geht es um die Unterstützung des Wertpapiergeschäfts in den Banken. Auch in diesem Projekt stellte der Anwendungsbereich sehr komplexe Anforderungen an eine software-technische Unterstützung. Daher wurde großer Wert auf die Analyse und einen guten fachlichen Entwurf gelegt. Dazu wurde die in [Grycan 93] beschriebene dokumentenorientierte Analyse- und Entwurfstechnik mit Szenarien, Glossar und Systemvisionen gewählt. In Ergänzung zu den Systemvisionen wurden einige Oberflächenprototypen mit dem Interface-Builder PARTS erstellt. Diese wurden zunächst als Labormuster in der internen Diskussion um Werkzeuge und Materialien verwendet. Bei der Auswertung von Szenarien und Systemvisionen wurden sie allerdings auch schon einzelnen Bankenvertretern als Demonstrationsprototypen gezeigt. Schließlich dienten die mit PARTS konstruierten Oberflächen als grafisches Material, um Systemvisionen zu erstellen.

PARTS ist ein in Smalltalk/V geschriebener Interface-Builder – verfügbar unter Windows und OS/2 – der ein Laufzeitmodul erzeugen kann. PARTS unterstützt unter OS/2 die gesamte Funktionalität des Presentation Managers, was insofern nicht unproblematisch ist, als daß damit Demonstrationsprototypen gebaut werden kännen, die mehr an Handhabungsmäglichkeiten zeigen, als sich mit dem eingeschränkten Mäglichkeiten der im Projekt verwendeten CommonView-Bibliothek realisieren läßt.

Erfahrungen der Entwickler

CASE/PM hat sich aus Sicht der Entwickler weniger zum Prototyping als mehr zur Einarbeitung in CommonView und Presentation Manager geeignet. Zwar ist das Werkzeug einfach zu handhaben, aber der angebotene Leistungsumfang (in der 1991 verfügbaren Version) entsprach nicht den Anforderungen des Teams. Da CASE/PM ein einziges großes Programmskelett generiert, konnten nur Wegwerf-Prototypen konstruiert werden. Schon kleinere €nderungen an der Oberfläche machten eine Neuimplementation des jeweiligen Prototyps notwendig. Zudem waren die Entwickler gezwungen, das Programmskelett manuell in kleinere Klassen zu zerlegen, um arbeitsteilig weiterentwickeln zu kännen. Nachdem die Entwickler einmal im Umgang mit CommonView geübt waren, wurde deshalb auf den Einsatz von CASE/PM ganz verzichtet.

Der Einsatz von PARTS für Demonstrationsprototypen war technisch wesentlich problemloser, da er in seiner Struktur dem Interace-Builder von NeXT sehr ähnlich ist. Er unterstützt somit nicht nur den Bau der Benutzungsoberfläche, sondern auch die Konstruktion der fachlichen Komponente. Die Einarbeitung wurde ebenfalls als relativ einfach bezeichnet. Durch die deutliche Trennung von Prototyping-Werkzeug (in Smalltalk) und Zielsystem-Umgebung (in C++) gab es keine Probleme mit der potentiellen Weiterverwendung von Demonstrationsprototypen – dies stand nie zur Diskussion. Problematisch war aus Sicht der Entwickler nur der hähere Funktionsumfang von PARTS gegenüber der CommonView-Bibliothek. Da PARTS auch von Nicht-Entwicklern im Team eingesetzt wurde, bestand die Gefahr, daß Oberflächenprototypen mehr an Handhabung zeigten, als hinterher umzusetzen war.

Die Verwendung von PARTS, um Systemvisionen – d.h. Entwurfsdokumente – zu erstellen, wurde als etwas aufwendig aber gegenüber anderen Mäglichkeiten als befriedigend eingeschätzt. So konnten in diesen Dokumenten nicht nur Textbeschreibungen, sondern auch gute grafische Entwürfe der Oberflächen künftiger Prototypen dargestellt werden.

Unsere Einschätzung

Die beiden in diesem Bankenprojekt eingesetzten Interface-Builder repräsentieren das Spektrum dessen, was heute gängig in industriellen Entwicklungsprojekten eingesetzt wird. Beide Systeme erlauben, im vorgegebenen Rahmen Oberflächenlayouts zu gestalten und diese mit elementaren Handhabungsformen zu versehen. Wir sehen, daß der Bruch dort entsteht, wo Anwendungsfunktionalität

hinzugefügt werden soll. Bei vielen Interface-Buildern bedeutet dies, daß in der Zielsprache programmiert werden muß – im Beispiel CASE/PM also C++. Die Fallstudie zeigt, daß dieser Programmieraufwand für Demonstrationsprototypen kaum investiert wird. Erschwerend kam hinzu, daß CASE/PM in der verwendeten Fassung ein einziges komplexes Programmskelett erzeugte, was weder für das Prototyping noch für die arbeitsteilige und zyklische Software-Entwicklung tragbar war.

PARTS hat unter diesem Aspekt große Vorteile. Allerdings steht in diesem Projekt die Programmiersprache (Smalltalk) einer evolutionären Weiterentwicklung des Prototyps vielfach im Wege. So stellt sich auch für PARTS die Frage, wieviel Anwendungslogik die Entwickler in einen nicht weiterverwendbaren Prototyp einprogrammieren wollen. Im vorliegenden Fall machte sich zudem bei beiden Interface-Buildern die Differenz in den Gestaltungsmäglichkeiten gegenüber der Zielumgebung bemerkbar. CASE/PM schränkte die Entwickler schon beim Layout zu stark ein. Daher konnte der funktionale Prototyp nur mit einer "manuell" erweiterten Oberfläche gebaut werden. Demgegenüber stellte PARTS für das Nachfolgeprojekt mehr an Handhabungs- und Layoutmäglichkeiten zur Verfügung, als die verwendete Fensterbibliothek zuläßt. Daher mußte ständig überprüft werden, ob die so konstruierten Oberflächenprototypen noch in den Projektrahmen passen und keine falschen Vorstellungen erzeugen.

Schließlich wollen wir noch auf zwei Aspekte eingehen, die als "Abfallprodukt" des Prototyping mit Interface-Buildern in diesem Projekt auffällig sind. Die Oberflächenlayouts werden als Grafiken in Entwicklungsdokumenten verwendet. Dies ist kein Einzelfall. Wir kennen Projekte, in denen solche Entwürfe nicht nur als Illustrationen in Dokumenten, sondern auch als Folienvorlagen für Systempräsentationen und Abstimmungsdiskussionen mit den Anwendern verwendet werden. Tendenziell erweist sich aber die parallele Fortschreibung von Oberflächenentwürfen und Textillustrationen als sehr aufwendig, da die Illustrationen in den Texten nicht mehr verändert werden kännen und daher stets der Weg über den Interface-Builder und einen Konversionsschritt (z.B. Bildschirmabzug) gegangen werden muß.

Der zweite Aspekt ist die Verwendung von Interface-Buildern als Lerninstrument, wenn neue Fenstersysteme eingesetzt werden. Dies unterstreicht sehr schän unsere These, daß Software-Entwicklung ein Lernprozeß aller beteiligten Gruppen ist, der durch Prototyping gefärdert wird. Denn in diesem Fall haben die Entwickler den Oberflächenprototyp als Labormuster genutzt, um sich die Arbeitsweise des Fenstersystems zu verdeutlichen.

Hauptsächlich werden Interface-Builder aber in diesem und einigen uns bekannten ähnlichen Projekten zum Bau von Demonstrationsprototypen oder reinen Oberflächenprototypen eingesetzt. Dazu eignen sie sich sehr gut. Eine kontinuierliche Weiterentwicklung über Prototypen im engeren Sinne zu Pilotsystemen und dem Anwendungssystem findet selten mit ihrer Hilfe statt. Dafür sehen wir zwei Gründe:

- Interface-Builder sind durch ihre Sprache oder ihre Gestaltungsmäglichkeiten zu eingeschränkt.
- Interface-Builder liefern Ergebnisse, die durch ihre Software-Architektur nicht in den Rahmen eines Entwicklungsprojektes passen. Für gräßere und komplexe Anwendungen sind weder Programmskelette noch ein reiner Callback-Mechanismus geeignet.

5.2.4 Auswertung der Werkzeuggruppe

Aufgrund der stark unterschiedlichen Ausprägungen von Interface-Buildern ist es nahezu unmäglich, allgemeingültige Aussagen für deren Einsatz beim Oberflächen-Prototyping zu machen. Insgesamt konnten wir feststellen, daß sowohl den Bau von Demonstrations- und Akquisitions-Prototypen als auch das evolutionäre Prototyping von Pilotsystemen durch Interface-Builder gut unterstützt werden, wenn einige wichtige Voraussetzungen von den Werkzeugen erfüllt werden.

Abhängig von Hardware-Umgebung und Betriebssystem ist das Angebot an guten Interface-Buildern sehr unterschiedlich. Die Bewertung der Entwickler aus der PC-Welt fällt gegenüber der Bewertung der UNIX-basierten Entwicklungsprojekte deutlich ab. Ausschlaggebend dafür ist nach unserer Meinung, daß im Workstation-Bereich auf der Basis von UNIX wesentlich mehr Werkzeugentwicklungen mit entsprechend gräßerem Aufwand gemacht wurden. Die entstandenen Werkzeuge sind mächtiger und besser für gräßere Software-Entwicklungen geeignet als Werkzeuge auf PC-Basis.

Eine an der Benutzungsoberfläche verwendbare, interpretierbare Sprache, mit der das Oberflächenverhalten definiert werden kann, ist eine unabdingbare Notwendigkeit, wenn Prototypen jeglicher Art erstellt werden sollen, selbst für reine Akquisitions- und Demonstrationsprototypen. Ohne eine solche Mäglichkeit wird Prototyping durch die dann notwendigen längeren †bersetzungs- und Bindevorgänge sehr mühsam, wenn nicht unmäglich. Systeme, die die Mäglichkeit anbieten, rasch vom Entwurf zur Simulation einer Oberfläche zu wechseln, erlauben, daß Prototypen in einem sehr kurzen Design und Evaluations-Zyklus erstellt werden kännen.

Interface-Builder, die prinzipiell nur Source-Code erzeugen, der später mit der fachlichen Komponente zusammen übersetzt und gebunden werden muß, sind für das Prototyping von Benutzungsoberflächen nur sehr eingeschränkt geeignet. Man benätigt zusätzlich zu dieser Eigenschaft die Mäglichkeit, die Interaktion mit den Oberflächen zu simulieren und zu evaluieren.

Wie Abschnitt 5.2.3 gezeigt hat, sind Systeme, bei denen Programmskelette entstehen, in die anschließend manuell Anwendungsfunktionalität integriert werden muß, nur sehr bedingt tauglich. Evolutionäres Prototyping ist mit solchen Systemen kaum durchführbar, da jeder Entwicklungszyklus mit umfangreichen Editiervorgängen verbunden ist. Als absolutes Minimum muß vorausgesetzt werden, daß zwei unabhängige Programm-Module (Oberfläche und fachliche

Komponente) entstehen, die ohne weitere Modifikationen übersetzt und gebunden werden kännen.

Auch die Mäglichkeit, den Interface-Builder und seine Oberflächenelemente während der Benutzung interaktiv zu erweitern, ist wichtig für den Einsatz eines solchen Werkzeugs im Prototyping-Prozeß. Die Mäglichkeit, Templates oder Klassen von ähnlichen Objekten zu definieren, ist unabdingbar, da sonst der Aufwand für Modifikationen an gräßeren Systemen (mit vielen Oberflächenelementen) unkalkulierbar wachsen kann, vor allem, wenn diese Modifikationen erst spät im Prototyping-Prozeß durchgeführt werden sollen.

Nach unserer Einschätzung läßt sich der überwiegende Teil aller Benutzungsoberflächen mit einer klar umrissenen Menge unterschiedlicher Oberflächenelemente erstellen, ohne daß die Werkzeuge erweitert werden müssen. Man muß jedoch abwägen, ob die Komplexität eines mächtigen Werkzeugs oder die Komplexität des Erweiterungsprozesses sich in der Arbeitsweise der Oberflächen-Designer stärker stärend bemerkbar macht.

Der vergleichsweise hohe Lernaufwand, der bei der Einführung vieler Interface-Builder zu erbringen ist, kann für den Prototyping-Einsatz dieser Werkzeuge stärend sein. Im Grunde gibt es dabei nur zwei Gruppen: auf der einen Seite stehen die einfach zu bedienenden Systeme, deren Mächtigkeit für den Bau von Prototypen nicht ausreicht, wenn diese mehr als das Layout der Oberfläche zeigen sollen. Auf der anderen Seite sind mächtige Werkzeuge zum Entwurf von Layout und Verhalten, die hervorragend für das Oberflächen-Prototyping geeignet sind, deren Benutzung jedoch (mehr oder weniger mühsam) erlernt werden muß.

In den von uns betrachteten Fällen war der Lernaufwand für die Entwickler nicht wirklich von Bedeutung, da sie entweder selbst an der Entwicklung der Werkzeuge beteiligt oder Experten in der Benutzung ähnlicher Werkzeuge waren. Dies läßt sich aber mit Sicherheit nicht verallgemeinern.

Der in Abschnitt 5.2.3 beschriebene positive Nebeneffekt der Nutzung von Interface-Buildern, nämlich bei der Einarbeitung in die Fenstersystemarchitektur und -programmierung, sollte nach unserer Auffassung nicht überbewertet werden. Die Tatsache, daß dies positiv bewertet wurde, zeigt nur, daß das verwendete System nicht hinreichend mächtig war. Die Eigenheiten der Fenstersysteme sollen durch Interface-Builder vor dem Benutzer verborgen bleiben, ohne die Entwurfsmäglichkeiten zu sehr einzuschränken. Die Qualität von Interface-Buildern im Prototyping-Prozeß zeigt sich nicht zuletzt daran, ob sie dem Benutzer die Mäglichkeit geben, komplexe Oberflächen zu entwerfen, ohne daß ihm die Komplexität der Umgebung bewußt zu werden braucht.

5.3 Projekte mit 4. Generationssystemen

4. Generationssysteme eignen sich zur Entwicklung von Prototypen für Informationssysteme auf Basis von (quasi-) relationalen Datenbanken. Die Prototypen lassen sich in einem evolutionären Prozeß zum Zielsystem weiterentwickeln.

In diesem Abschnitt berichten wir über zwei Projekte, bei denen Prototypen mit 4. Generationssystemen erstellt wurden: Die Entwicklung eines Projektabwicklungssystems wurde mit 4th Dimension durchgeführt, ein Bank-Arbeitsplatz mit Windows/4GL.

5.3.1 Ein System zur Vorprojektkalkulation und Projektabwicklung

Das untersuchte Projekt

Der Bereich Prozeßautomatisierung einer Firma, die Industrieanlagen baut, suchte schon seit langem nach einem System für die Projektabwicklung und Vorprojektkalkulation, im weiteren PuV-System genannt. Ein für diesen Bereich passendes System konnte nicht gefunden werden. Die Adaptierung von ähnlichen Systemen (auf Großrechnerbasis) hätte nach Schätzung ca. ein Jahr erfordert. Aus der Zusammenarbeit der Firma mit der Software Engineering Abteilung einer Hochschule entstand die Idee, das PuV-System auf Arbeitsplatzrechnern zu realisieren.

Aufgabe des PuV-Systems ist, bei der Planung und Durchführung von Projekten die notwendigen Daten zu erfassen und so aufzubereiten, daß ein solches Projekt vorab kalkuliert und nach erfolgreichem Vertragsabschluß durchgeführt werden kann. Dabei mündet die Vorkalkulation in einem Angebot. Bei der Projektdurchführung werden die Verwaltung von Projekten auf der Basis der Vorprojekte und deren Revisionen unterstützt. Gegenstand von Planung und Durchführung sind die Materialwirtschaft einschließlich gewünschter Hardware- und Software-Vorgaben, der Einsatz der verfügbaren Mitarbeiter und die Kostenkalkulation. Dabei lag der Schwerpunkt des PuV-Systems auf einer speziell auf die Benutzer zugeschnittenen Informationserfassung und unterschiedlichen Auskunftsmäglichkeiten. Das System hat geringe algorithmische Komplexität. Die Erfassung von externen Daten wie Grafiken oder Spreadsheets ist nicht notwendig.

Vertragsbestandteile für die Entwicklung eines solchen PuV-Systems waren der Kosten- und Zeitrahmen sowie die allgemeine Aufgabenstellung für das Entwicklungsprojekt. Rechnerauswahl, Werkzeugeinsatz und Vorgehensweise wurden zwischen Entwicklern und Anwendern abgestimmt, wobei die Vorschläge von den Entwicklern kamen.

Benutzte Werkzeuge

Entwickelt werden sollte das PuV-System als Informationssystem auf Basis einer relationalen Datenbank. Nach der Evaluierung der auf dem Markt erhältlichen 4G-Systeme, entschieden sich die Entwickler, 4th Dimension auf Apple Macintosh Rechnern einzusetzen. Die einzelnen Rechner sind in ein LAN eingebunden. Eine externe Datenbank ist bis jetzt noch nicht im Einsatz.

Der Entwicklungsprozeß

Die Anzahl der am Entwicklungsprozeß beteiligten Personen wurde von vornherein bewußt klein gehalten, um die Arbeitsfähigkeit der Gruppe nicht einzuschränken. Neben drei Benutzern war auf Seite der Anwender ein Projektleiter und gelegentlich ein Vertreter des obersten Managements beteiligt. Das Entwicklerteam bestand im wesentlichen aus zwei Personen.

Zunächst wurde vom Auftraggeber ein ca. 40-seitiges Pflichtenheft mit einem durchformulierten Beispiel erstellt. Dieses schien auch den Entwicklern ausreichend zu sein, um das Projekt zu beginnen. Anhand dieses Pflichtenheftes wurde in drei Wochen ein erster Prototyp konstruiert und dem Auftraggeber vorgestellt. Dabei stellten die beteiligten Personengruppen fest, daß das Pflichtenheft falsch interpretiert bzw. nicht detailliert genug geschrieben war. Der Prototyp war bezüglich der Funktionalität noch ungenügend und auch in der Gestaltung der Oberfläche gab es noch große Abweichungen. Daher diente er nur als Demonstrationsprototyp für die generelle Entwicklungsrichtung. Auch der zweite funktionale Prototyp wurde komplett von seinem Nachfolger abgeläst. Erst der dritte Prototyp bildete die Grundlage für den weiteren evolutionären Prozeß, der zunächst ein halbes Jahr dauerte. Parallel zu den Prototypen wurde das Datenmodell entworfen, das ebenfalls evolutionär weiterentwickelt wurde. In dieser Zeit wurde etwa im zweiwächigen Rhythmus eine neue Version des Prototyps erstellt und mit Anwendern und Benutzern diskutiert. Nach einem halben Jahr wurde ein Pilotsystem bei drei Benutzern installiert. Dieses umfaßte allerdings nur Vorprojekt- und Mitarbeiterverwaltung. Ein Pilotsystem, das die gesamte Funktionalität abdeckte, wurde nach einem weiteren halben Jahr fertiggestellt und beim Kunden installiert. Für den weiteren Ausbau ist geplant, eine Laptopversion zu erstellen, in der nur die Daten einzelner für den jeweiligen Bearbeiter interessanten Vorprojekte übernommen werden.

Zwischen den beteiligten Personengruppen wurde vereinbart, nach einer Laufzeit von ca. einem Jahr ein technisches und fachliches Redesign einzuplanen. Dadurch sollte sowohl die Akzeptanz des Systems beim Anwender als auch die software-technische Architektur des Systems verbessert werden.

Die verschiedenen Prototypen wurden bzgl. Entwurf und Konstruktion in folgendem Prozeß bewertet. Zunächst fand ein internes Review im Entwicklerteam statt. Danach wurden die Prototypen von einem Team, das sich aus den Entwicklern, drei Benutzern und einem Vertreter des mittleren Anwendermanagements (Projektleiter) zusammensetzte, evaluiert. Im Rhythmus von zwei Monaten wurde

das Projekt mit dem Top-Management des Auftraggebers abgestimmt. Anfangs gab es Reibungsverluste zwischen Entwicklern und den anderen beteiligten Gruppen, wobei die Rollentrennung in "Anbieter" und "Käufer" im Vordergrund stand. Dazu kam, daß die Anwender Grenzen und Mäglichkeiten des neuen Systems noch nicht einschätzen konnten und oft zu detaillierte Forderungen stellten (ein neuer Prototyp für jeden mäglichen neuen Knopf). Mit dem gegenseitig wachsenden Verständnis änderte sich dieses Verhältnis der Gruppen zueinander. Heute besteht der Eindruck eines "integrierten Teams" mit unterschiedlichen Qualifikationen und Aufgaben. Die Benutzer fühlen sich voll in den Entwicklungsprozeß einbezogen und identifizieren sich mit dem System. Zudem werden neue Anforderungen an das PuV-System auf der Anwenderseite heute genauer auf ihre Nützlichkeit untersucht. Obwohl die letztliche Entscheidung über Entwurfsalternativen beim Anwendermanagement liegt, stellen sich in der Diskussion über solche Alternativen anhand von Prototypen die Forderungen der Benutzer meist als überzeugend heraus.

Die Entwickler stellten fest, daß nach ca. einem Jahr verstärkt konstruktive Vorschläge von der Benutzerseite kamen. Es war aber auch festzustellen, daß in der reinen Prototyping-Phase Systemteile gefordert wurden, die dann im Piloteinsatz nicht verwendet wurden. Insgesamt verlief der Prozeß von der anfänglichen Diskussion der Handhabung und Funktionalität zur Diskussion des Datenmodells und der geeigneten Präsentation durch die Oberfläche, wobei dies natürlich eng mit der Funktionalität gekoppelt ist.

Sämtliche Prototypen wurden mit 4D entwickelt. Die Benutzungsoberfläche, die vom Entwicklungswerkzeug generiert wurde, wurde durch eine eigene Oberfläche ersetzt. Für die Gestaltung der Oberfläche wurde ein Spezialist für Software-Ergonomie einbezogen.

Während des Prototyping-Prozesses wurde ein Projekttagebuch geführt, in dem alle Besprechungen zwischen den Benutzern und den Entwicklern (Prototyping-Sessions) protokolliert sowie davon abgeleitete Entwurfsentscheidungen festgehalten sind. Die kurze Systemdokumentation enthält eine Beschreibung der Systemarchitektur, das Datenmodell, Programmierkonventionen sowie Erklärungen von komplexeren Programmteilen. Neben dieser Dokumentation waren vor allem auch Werkzeuge der Entwicklungsumgebung (Cross-Referencer und Struktureditor) sehr hilfreich, um eine effiziente Wartung des Systems durchzuführen.

Nach einem Jahr wurde begonnen, das PuV-System zu optimieren. Dabei konnten 4/5 der von den Benutzern und Anwendern genannten Probleme einfach behoben werden. Der Rest mußte zunächst durch Messungen genauer lokalisiert werden.

Das System wird heute im Top-Management durchweg positiv bewertet, was unter anderem auch an den guten Reportmäglichkeiten von 4D liegt, die auch Geschäftsgraphiken einschließen.

Der Einsatz des PuV-Systems innerhalb der Firma führte zu einer Reorganisation, da das System einheitlich für die gesamte Abteilung eingesetzt wird. Auf diesem Wege wurde auch eine organisatorische Vereinheitlichung von Projektplanung und -

abwicklung erreicht (was das Management bereits als internes Ziel angestrebt hatte). Zudem änderte sich die Hardware-Ausstattung von einer Monokultur zu einer vernetzten heterogenen Netzarchitektur von Arbeitsplatzrechnern mit grafischer Benutzungsoberfläche und Großrechnern.

Der Einsatz und die Weiterentwicklung des PuV-Systems wurde vertraglich zwischen den beteiligten Parteien auf mindestens fünf Jahre festgelegt. Dabei verpflichtet sich der Hersteller, einen Arbeitsaufwand von mindestens drei Tagen im Monat für die Wartung bereitzustellen.

Die konstruierten Prototypen

Prototyp I: Er wurde anhand eines Pflichtenheftes entwickelt, entsprach dann aber weder von der Handhabung noch von der Funktionalität den Vorstellungen des Auftraggebers. Obwohl er nicht so geplant war, diente er ausschließlich als Demonstrationsprototyp. Er modellierte einen Ausschnitt des Systems (Vorprojekt) und besaß bereits vertikale Elemente.

Prototyp II: Er war bereits eine bessere Näherung zwischen den Vorstellungen der Benutzer und dem Verständnis der Entwickler. Da seine Auswertung aber noch zahlreiche Mängel zeigte, erwies es sich aufgrund des eingesetzten Werkzeugs als einfacher, diesen Prototyp wegzuwerfen statt ihn evolutionär weiterzuentwickeln.

Prototyp III: Er wurde ebenfalls komplett neuentwickelt und kann als Prototyp im engeren Sinne bezeichnet werden. Er zeigte bereits große Teile der Oberfläche und der Funktionalität. Auf seiner Basis wurden evolutionär in kurzen Abständen (ca. zwei bis drei Wochen) eine Folge von weiteren Prototypen entwickelt und mit den Benutzern diskutiert. Dabei wechselten sich die Modellierung des Datenmodells und der Ausbau der Funktionalität zyklisch ab. Nach insgesamt einem halben Jahr stabilisierten sich die Anforderungen an die Prototypen.

Pilotsystem I: Es wurde bei drei Benutzern mit jeweils unterschiedlichen Arbeitsaufgaben eingesetzt. Das Pilotsystem unterstützte allerdings nur Vorprojektabwicklung und Mitarbeiterverwaltung. Es wurde laufend weiterentwickelt, da ständig Anregungen der Benutzer besonders im Bezug auf die Oberflächengestaltung kamen. Diese Abstimmung der Oberfläche auf die Anforderungen der einzelnen Benutzer führte zu einer sehr hohen Akzeptanz des Systems.

Prototyp IV: Er wurde parallel zum Pilotsystem I entwickelt und modellierte die eigentliche Projektverwaltung.

Pilotsystem II: Es umfaßt die gesamte gewünschte Funktionalität. Der Einsatz dieses Pilotsystems führte nach einer Projektlaufzeit von insgesamt einem Jahr zu gräßeren €nderungswünschen, die hauptsächlich in der gewünschten Trennung zwischen der Verwaltung von Kunden und Mitarbeitern begründet ist.

Erfahrungen der Entwickler

Das 4G-System 4th Dimension wurde von den Entwicklern als praxisgerecht empfunden. Zudem unterstützt es den evolutionären Prototyping-Prozeß gut. Allerdings muß die integrierte Programmiersprache mit einem gewissen Maß an Disziplin eingesetzt werden. Positiv wurde die Mäglichkeit bewertet, externe Routinen einzubinden sowie ein Laufzeitsystem abgekoppelt von der Entwicklungsumgebung erzeugen zu kännen.

Unsere Einschätzung

Dieses Projekt kann unter verschiedenen Gesichtspunkten als typisches Prototyping-Projekt gesehen werden. Hier wäre zunächst das Pflichtenheft als Ausgangspunkt des ersten Prototyps. Dabei wird deutlich, daß es schwierig ist, interaktive Systeme ausreichend und für alle beteiligten Personengruppen verständlich in Pflichtenheften zu beschreiben. Gleichzeitig muß aber festgestellt werden, daß trotz dieser mittlerweile verbreiteten Einsicht das Pflichtenheft eine große Bedeutung für Vertragsabschlüsse und Projektinitiierung hat. Wir folgern daraus, daß neue Dokumenttypen und eine andere Art der Vertragsgestaltung für Prototyping-Projekte erforderlich sind.

Bemerkenswert ist weiterhin, daß, obwohl die evolutionäre Weiterentwicklung der Prototypen zum Anwendungssystem geplant war, die ersten beiden Prototypen nur als Diskussionsbasis für die beteiligten Personengruppen dienten. Hier wurde die richtige Konsequenz aus der Einsicht gezogen, daß die ersten Prototypen in einem komplexen neuen Anwendungsbereich noch zu weit von den Vorstellungen der Anwender entfernt waren, als daß sie technische Grundlage der weiteren Entwicklung hätten sein kännen. Erfahrungen aus anderen Projekten belegen, daß der Versuch, durch schrittweise Nachbesserungen zu einem tragfähigen Anwendungssystem zu kommen, in solchen Fällen oft zu fachlich und software-technisch unakzeptablen Läsungen führt, die ein Projekt insgesamt zum Scheitern bringen kännen.

Die berichteten Erfahrungen über die Einstellungsänderungen der beteiligten Personengruppen bei der Evaluierung der verschieden Prototypen scheinen uns ebenfalls typisch zu sein. Die wachsende Integration der verschiedenen Gruppen in ein kooperatives und produktives Team läßt sich nur langsam erreichen. Hier wird besonders die geänderte Rolle der Anwender deutlich, die nicht mehr vorrangig Käufer oder Informationslieferanten, sondern konzeptionell mitwirkende Mitglieder des Entwicklungsteams wurden. Dieses kooperative Team ermäglichte es dann auch, das Anwendungssystem evolutionär zu entwickeln. Allerdings stellt sich mit Blick auf den Beteiligungsaspekt die Frage, wie gräßere Anwendergruppen effektiv an einem Entwicklungsprozeß teilnehmen kännen. Hier und aus anderen Projekten läßt sich feststellen, daß kleine Entwicklergruppen von etwa 10 bis max. 15 Personen (die Anwender eingerechnet) optimal produktiv sind.

Ein 4G-System vom Typ 4D scheint gut geeignet, um evolutionäres Prototyping bei der Konstruktion von Informationssystemen einzusetzen. Charakteristisch ist

hierbei, daß eine getrennte Diskussion der Oberfläche nicht sinnvoll und mäglich war. Als Tendenz zeichnet sich ab, daß bei komplexen Anwendungen die Gestaltung der Benutzungsoberfläche eher von der fachlichen Komponente der Anwendung her erfolgt, als daß sich umgekehrt die fachliche Komponente von der Oberflächengestaltung her entwickeln läßt. Da 4D eine direkte Verzahnung von Datenbankschema und Oberflächengestaltung ermäglicht, kann dieser Zusammenhang leicht aufrechterhalten werden. Dazu kommt, daß der rasche Wechsel zwischen Entwicklungsmodus (für Schema und Oberfläche) und experimenteller Erprobung für Prototyping-Prozesse sehr färderlich ist.

Schließlich scheint uns bemerkenswert, daß nach längerem Piloteinsatz bewußt ein tiefgreifendes Redesign eingeplant und umgesetzt wurde. Dadurch wird vermieden, daß sich durch die schrittweise Entwicklung eines Systems ein Strukturverlust einstellt und daß tiefergreifende €nderungswünsche ständig aufgeschoben werden.

5.3.2 Ein Arbeitsplatz für Konzernbetreuer

Das untersuchte Projekt

Die Qualität der Beratung im Bankgewerbe ist stark davon abhängig, wie schnell aktuelle Informationen verfügbar sind. Da diese von unterschiedlichen Anbietern bereitgestellt werden, muß ein Berater mehrere Informationssysteme mit verschiedenen Benutzungsoberflächen und oft auch unterschiedlicher Hardware benutzen. Neben dem geringen Komfort (z.B. mehrere Terminals auf dem Schreibtisch) kommt vor allem der Nachteil zum Tragen, daß einmal eingegebene Daten nicht oder nur mit großem Aufwand von anderen Informationssystemen weiterverarbeitet werden kännen.

Ziel dieses Projektes war es, Erkenntnisse über die Gestaltung und Realisierung eines elektronischen Bankarbeitsplatzes zu gewinnen. Dieser sollte so gestaltet sein, daß alle benätigten Dienste (Bürofunktionen, Hilfsmittel zur Kreditprotokollierung, Auswertungen zur Berechnung der Kundenrentabilität, Zugriff zu aktuellen Kunden- und Finanzmarktinformationen) an einem einzigen Terminal und über eine einheitliche Benutzungsoberfläche zugänglich sind.

Als Anwendungsgebiet wurde die Tätigkeit eines Konzernbetreuers ausgewählt. An einem solchen Arbeitsplatz stehen zur Zeit noch vier verschiedene Terminals. Zwischen den dort laufenden Anwendungen bestehen keine oder nur unbefriedigende †bergänge.

Am Projekt waren die künftigen Benutzer, die Entwickler und, zu Review- und Genehmigungszwecken, auch der Auftraggeber beteiligt.

Die Gründe für die Entwicklung verschiedener Prototypen waren vielschichtig:

- Ein vorangehendes Projekt mit ähnlicher Aufgabenstellung scheiterte mit dem Ansatz der systematischen Funktionsanalyse.

- Es mußten Verständigungsprobleme zwischen Benutzern und Entwicklern beseitigt werden (die Oberfläche der verwendeten Werkzeuge waren den meisten Benutzern unbekannt).
- Die Ideen und Visionen der Entwickler mußten schnell umgesetzt werden kännen, so daß sie von den Benutzern beurteilt werden kännen.
- Da es sich um ein Innovationsprojekt handelt und das gesamte Vorgehen neu war, sollte auch das (vorläufige) Endprodukt, das Pilotsystem, mit Prototyping-Ansätzen entwickelt werden.

Die Prototypen wurden in den verschiedenen Projektphasen entsprechend ihrer Zielsetzung mit einem jeweils passenden Werkzeug entwickelt.

Das Ziel des Auftraggebers war, eine gemeinsame Benutzungsoberfläche für eine gräßere Anzahl bestehender Anwendungen entwickeln zu lassen. Im Laufe der Projektplanung und durch Diskussion mit den Entwicklern entschloß man sich jedoch, daß nur ein komplettes Produkt, d.h. Benutzungsoberfläche und die zugehärige Funktionalität, die gestellte Aufgabe befriedigend läsen kann.

Benutzte Werkzeuge

Neben fachlichen Läsungen sollte das Projekt auch dazu dienen, Erfahrungen mit verschiedenen Prototyping-Werkzeugen zu gewinnen. So kamen für die ersten Prototypen die Werkzeuge Intermedia, SuperCard und ET++ zum Einsatz. Weitere Entwicklungswerkzeuge waren DB-Designer und DEFT (zur Erstellung von Entity-Relationship-Diagrammen und des Data Dictionaries).

Das Pilotsystem wurde mit Windows/4GL von Ingres unter UNIX entwickelt. †ber externe Schnittstellen sind folgende Informationsdienste und Werkzeuge zugänglich:

- X-Desktop
- Framemaker (zur Textverarbeitung)
- X-Claim (zur Tabellenkalkulation)
- email, voicemail
- Radio (z.B. zum Abrufen der letzten Nachrichten)
- IPS (Retrieval System mit Anschluß an Handelsblatt, Handelsamtsblatt per Diskette und Direktanschluß an Dow Jones)

Der Entwicklungsprozeß

Das Projekt hatte eine Laufzeit von vier Jahren und war in sechs Phasen unterteilt:

Evaluierung des Problemumfelds:

Ein Benutzer und zwei Entwickler arbeiteten sich in die fachliche Thematik ein, indem sie die tägliche Arbeit der Benutzer beobachteten. Ergebnis war der Konzeptentwurf für das Projekt, der als Entscheidungsvorlage für das Management diente.

Zusammenwachsen eines interdisziplinären Projektteams:

Das Team bestand aus Benutzern, Entwicklern und einem externen Unternehmensberater. Die zwei bis zehn Mitglieder benätigten ein ganzes Jahr, um zu einem gefestigten Projektteam zusammenzuwachsen und ein gemeinsames Verständnis des Problems zu haben. Die Dauer dieses Abschnitts war für alle Beteiligten erstaunlich hoch. Sie war geprägt durch systematische Benutzerbefragungen, in die auch die Kunden der Bankberater einbezogen wurden. Bei den ersten technischen Diskussionen ergaben sich z.T. heftige Meinungsverschiedenheiten über das weitere Vorgehen, was wesentlich zur Dauer dieser Phase beigetragen hat, für den Erfolg des Gesamtprojekts aber unerläßlich war.

Konzeptphase:

Aufbauend auf dem nun gefestigten Problemverständnis wurde in nur drei Monaten das endgültige Konzept erstellt. Beteiligt daran waren fünf Benutzer und fünf Entwickler. Parallel dazu entstanden drei Prototypen mit Intermedia, SuperCard und ET++, die das Konzept maßgeblich beeinflußten. Zum Abschluß der Phase fiel die Entscheidung des Managements, daß der eigentliche Prototyp mit Windows/4GL gebaut wird.

Klärungsphase:

Nachdem das Konzept feststand, rückte jetzt die Realisierung in den Mittelpunkt des Interesses. Als Konsequenz daraus verringerte sich der Anteil der Benutzer im Projektteam deutlich: einem bis zwei Benutzern standen vier Entwickler gegenüber. Es entstanden mehrere Labormuster mit Windows/4GL, mit denen die zu verwendenden externen Schnittstellen untersucht wurden. Nach einem dreiviertel Jahr stand das Vorgehen für die Realisierung fest. Ursprünglich wurde das gesamte Problem als Dokumentenverwaltung betrachtet, das unter Verwendung von Hypertext-Ansätzen geläst werden kann. Inzwischen hat man jedoch erkannt, daß die Läsung aus einer Sammlung von Datenbankanwendungen besteht.

Realisierung:

Die Entwickler (2-12) benätigten dazu ein knappes Jahr. Die Benutzer standen als Berater und Auskunftspersonen zur Verfügung. Schwerpunkte der Realisierung waren Datenbankanwendungen, Schnittstellen zu externen Systemen, Dokumentenverwaltung und Systemaspekte im Hinblick auf die Pilotphase. Hier kam zum Tragen, daß die Realisierung von Teilen der Funktionalität optional war. Diese Phase konnte damit termingerecht abgeschlossen werden.

Der für ein halbes Jahr geplante Piloteinsatz des Windows/4GL-Prototyps begann am Tag nach Durchführung des Interviews. Ergebnisse liegen daher noch keine vor.

Die konstruierten Prototypen

Es wurden drei Arten von Prototypen erstellt:

1. Explorative Prototypen mit Intermedia und SuperCard (in der Konzeptphase)

Sie waren Teil der Spezifikation und dienten zur Demonstration und Diskussion mit den Benutzern. Die eingesetzten Werkzeuge wurden im wesentlichen zum Zeichnen verwendet. Intermedia, ein Werkzeug zur Verarbeitung von Hypertext-Dokumenten, wurde dabei nicht bestimmungsgemäß eingesetzt (s. auch Philosophieänderung in der Klärungsphase). In einem weiteren Projekt mit gleicher Aufgabenstellung käme dieses Werkzeug wegen seiner beschränkten Mäglichkeiten nicht mehr zum Einsatz, obwohl es sehr schnell zu erlernen ist (Aufwand < 1 Tag).

SuperCard wurde bestimmungsgemäß verwendet. Der im Vergleich zu Intermedia hähere Lernaufwand (einige Tage) ist durch die gräßere Mächtigkeit des Werkzeugs gerechtfertigt. Es wird im Wiederholungsfall erneut eingesetzt. Bei den Benutzern erlangten diese Prototypen eine hohe Akzeptanz, da sie beim Durchspielen einzelner Szenarien sofort die Handhabbarkeit sowie den mäglichen Nutzen des Prototyps erkennen konnten. Entsprechend ihrer Bestimmung wurde über diesen Prototypen nur eine Broschüre erstellt, aber keine technische Dokumentation geschrieben.

2. Ein explorativer und experimenteller Prototyp mit ET++

Dieser Prototyp, der ebenfalls in der Konzeptphase erstellt wurde, diente dazu, die Besitzstrukturen in Großkonzernen zu visualisieren sowie ihre Manipulation zu erproben. Es handelte sich dabei um einen grafischen Editor/Browser.

Die Entwicklung erfolgte mit dem ET++ Application-Framework. Der Entwickler des Prototyps hatte Schwierigkeiten, sich in einem vertretbaren Aufwand in das Framework einzuarbeiten. Ein nochmaliger Einsatz dieser Technologie würde deshalb nur unter günstigeren Zeitbedingungen oder bereits vorhandenen ET++-Kenntnissen erwogen.

3. Ein evolutionärer Prototyp mit Windows/4GL

Mit diesem Prototyp wurde das gefestigte Konzept realisiert. Er wurde als Prototyp im engeren Sinn entwickelt und bis zum einsetzbaren Pilotsystem weiterentwickelt. Die Entwicklung erfolgte unter ständiger Rücksprache mit den Benutzern, so daß auch die Oberfläche erst jetzt ihr endgültiges Gesicht bekam. Der Einsatz von Windows/4GL wurde von den Entwicklern positiv betrachtet.

Während der Pilotphase wird bei der Schulung und beim Einsatz des Systems beobachtet, inwieweit die Benutzer das System akzeptieren. Parallel zum Prototyp wurde die Entwicklungsdokumentation erstellt.

Erfahrungen der Entwickler

Der enge Kontakt zwischen Entwicklern und Benutzern führte bei den Entwicklern zu einer hohen Sensibilität für die fachlich Problematik. Die Entwicklung der Prototypen hat wesentlich dazu beigetragen.

Ein Hauptproblem des Projekts war die Teambildung bei den Entwicklern (siehe Phase II). Dies lag an der Neuigkeit des Vorgehens und der eingesetzten Werkzeuge sowie an der Spannung zwischen Forschungsauftrag und dem Ziel, ein einsetzbares Produkt zu erstellen. Man benätigte daher viel Zeit für die Diskussion der Methodik, des Entwicklungsstils sowie die Verteilung von sinnvollen Arbeitsgebieten für die einzelnen Entwickler. Für den Ablauf des Projekts war es sehr wichtig, in dieser Phase genügend Zeit zu haben.

Nach den prinzipiell positiven Erfahrungen mit Windows/4GL würden die Entwickler bei einem erneuten Projektstart das Pilotsystem versuchsweise mit dem DB-Kit von NeXT entwickeln. Sie versprechen sich davon eine hähere Entwicklungsgeschwindigkeit durch die enge Kopplung an die Datenbank und gräßere Flexibilität durch die bereits vorhandenen Werkzeuge zur Büroautomatisierung. So sind dort z.B. Werkzeuge wie Textverarbeitung, Tabellenkalkulation, Terminplaner verfügbar und über Schnittstellen in die Anwendung einzubinden. Eine Entscheidung zwischen Windows/4GL und NeXT kann aus heutiger Sicht nicht gefällt werden.

Unsere Einschätzung

Das Projekt nimmt mit seiner langen Laufzeit (vier Jahre) eine Sonderstellung ein. Dadurch bedingt konnten vier Werkzeuge eingesetzt werden, um die Prototypen zu entwickeln: Intermedia, SuperCard, ET++ und Windows/4GL. Mit Ausnahme von Intermedia wurden alle bestimmungsgemäß eingesetzt.

Mit den drei erstgenannten Werkzeugen wurden Prototypen zur Konzeptklärung erstellt. Wichtig war dabei die Mäglichkeit, die Oberfläche und ihr interaktives Verhalten schnell zu implementieren, so daß sie als Grundlage für die Diskussion mit den Benutzern dienen konnte. Zur Erstellung des Pilotsystems wurde Windows/4GL verwendet.

Auch hier kännen Oberflächen schnell und komfortabel entwickelt werden. Ein schnelles Wechseln vom Entwicklungs- in den Ausführungsmodus sowie die

Unterstützung der Teamarbeit durch eine entsprechende Versionenverwaltung begünstigen den Einsatz von Windows/4GL zum Prototyping. Die integrierte 4GL ermäglicht es, auch komplexe Codestücke zu strukturieren und damit gut wartbare Software zu entwickeln.

Hervorzuheben ist an diesem Projekt, daß das Ziel, eine gemeinsame Benutzungsoberfläche für bereits bestehende Anwendungen zu entwickeln, nur erreicht werden konnte, indem, entgegen der ursprünglichen Absicht, eine komplette Anwendung, d.h. Benutzungsoberfläche einschließlich der zugehärigen fachlichen Komponenten, entwickelt wurde.

5.3.3 Auswertung der Werkzeuggruppe

Bei den hier betrachteten Projekten wurden Prototypen entwickelt, weil neue Entwicklungsumgebungen eingeführt wurden.

Die Auswertung der untersuchten Projekte zeigt, daß 4G-Systeme gut geeignet sind, um Prototypen zu erstellen. Sie ermäglichen, daß Oberflächen schnell entworfen und verändert werden kännen und daß rasch zwischen Entwicklungs- und Ausführungsmodus hin und her gewechselt werden kann. Aufwendig ist es lediglich, das Datenbankschema zu ändern, da dies von den Werkzeugen kaum oder gar nicht unterstützt wird. Mit den ausgewählten 4G-Systemen konnten sämtliche Arten von Prototypen entwickelt und in einem evolutionären Prozeß zum Pilotsystem ausgebaut werden.

Aufgrund ihrer Eigenschaften sind jedoch eng und lose gekoppelte 4G-Systeme für unterschiedliche Einsatzgebiete prädestiniert:

Die Benutzungsoberfläche einer Anwendung, die mit Hilfe eines eng gekoppelten 4G-Systems entwickelt wird, kann aus der Definition des physischen Datenmodells generiert werden (siehe Kapitel 4.3.1). Damit ist die Verwendung eines eng gekoppelten 4G-Systems nur für solche Anwendungen geeignet, bei denen die Sicht der Benutzer auf die Daten mit dem physischen Datenmodell übereinstimmt.

In eng gekoppelten 4G-Systemen muß die Datenmanipulation nicht explizit programmiert werden. Das hat zur Folge, daß sich Prototypen schnell entwickeln lassen. Diese Eigenschaft begünstigt es, Demonstrationsprototypen zu erstellen. Sie kommt aber auch dann zum Tragen, wenn eine konzeptionelle €nderung durchgeführt werden soll, da sich ein neuer Prototyp leicht entwickeln läßt. So ist die Gefahr gering, daß am bestehenden Prototyp aus Kostengründen nur die im Rahmen des alten Konzepts mäglichen €nderungen durchgeführt werden. So wurden beim PuV-System mehrere Prototypen entwickelt, in die zwar die Erfahrung der vorhergehenden Prototypen, aber nicht dessen Programmcode eingeflossen sind.

Die Leistungsfähigkeit der eingesetzten Werkzeuge unterscheidet sich durch die zugrunde liegenden Basissysteme (Oberfläche, 4.GL, Datenbank). So bietet z.B. 4D keine Modularisierungskonzepte für den erstellten Code an. Soll der Prototyp evolutionär bis zum Zielsystem weiterentwickelt werden, so müssen bei der

Werkzeugauswahl neben der Eignung für das Prototyping auch diese Anforderungen berücksichtigt werden.

5.4 Application-Frameworks

Objektorientierte Application-Frameworks werden immer populärer, um funktionsfähige Prototypen zu erstellen, die anschließend evolutionär zum fertigen Zielsystem weiterentwickelt werden kännen.

Wir zeigen in diesem Abschnitt drei Fallstudien für Prototyping-Projekte, bei denen Application-Frameworks eingesetzt wurden. Bei allen Projekten war die Gestaltung der Benutzungsoberfläche von sehr großer Bedeutung.

Bei den ersten beiden Projekten wurden grafische Editoren entwickelt: ein Editor für SWIFT-Nachrichten und ein Editor, um technische Systeme zu modellieren. In beiden Projekten wurde mit dem Application-Framework Smalltalk gearbeitet. Die dritte Fallstudie beschreibt ein Projekt, bei dem ein Arbeitsplatz für einen SWAPS-Händler entwickelt wurde. Dazu wurde das Application-Framework ET++ eingesetzt.

5.4.1 Ein Editor für SWIFT-Nachrichten

Das untersuchte Projekt

Seit längerer Zeit wird ein Software-System bei Banken eingesetzt, mit dem SWIFT-Nachrichten sowohl innerhalb einer Bank versendet als auch zwischen Banken ausgetauscht werden kännen. SWIFT ist ein normiertes Nachrichtenformat für Transaktionen im Bankensektor. Das System ist eingeführt und weltweit im Einsatz. Es ist in Assembler realisiert.

Eine Entwicklungsabteilung des Herstellers gab zusammen mit der Marketing-abteilung den Auftrag, einen Teil des Systems zu verbessern. Das neu zu entwickelnde Subsystem ist eine Komponente, mit der definiert werden kann, wie SWIFT-Nachrichten zwischen den einzelnen Funktionen (z.B. Arbeitsplätzen) in einer Bank ablaufen sollen. Zudem sollte es mäglich sein, den Nachrichtenfluß zwischen den einzelnen Bankfunktionen (z.B. Eingang, Drucken, Verifizieren durch Vorgesetzten usw.) durch Bedingungen zu definieren, die als Entscheidungstabellen betrachtet werden kännen.

Diese Tätigkeit wurde bisher folgendermaßen durchgeführt: Es wurden auf Papier entsprechende Szenarien erstellt, die den Nachrichtenfluß zeigen und die Bedingungen enthalten. Diese Szenarien wurden dann direkt in Assemblermakros umgesetzt.

Das Projekt wurde initiiert, um festzustellen, ob die Banken bereit sind, dieses Produkt zu erwerben und einzusetzen. Die Marketingabteilung erteilte den Auftrag, die Komponente so zu realisieren, daß die Definition des Nachrichtenflußes interaktiv grafisch mit einem Werkzeug geschehen soll, das anschließend die entsprechenden Assemblerteile generiert. Weiterhin sollte es in der Lage sein,

bestehende Entscheidungstabellen einzulesen und zu visualisieren. Diese Komponente wird im weiteren als SNE (SWIFT-Nachrichten-Editor) bezeichnet.

Da es kein Vorbild für eine solche Anwendung gab, war von vorne herein geplant, innerhalb eines evolutionären Entwicklungsmodells Prototyping durchzuführen, damit ein Prototyp, der die Darstellungsform, die Handhabung und das Verhalten von SNE zeigt, den potentiellen Anwendern demonstriert werden kann. Die potentiellen Anwender sind die Banken, die bereits das existierende SWIFT-System im Einsatz haben.

Benutzte Werkzeuge

Es wurde entschieden, Smalltalk/VPM der Firma Digitalk unter OS/2 mit Presentation Manager auf einem PC einzusetzen. Die Entscheidung für Smalltalk/VPM hatte folgende Gründe:

- SNE sollte objektorientiert implementiert werden.
- Diese Version erlaubt, Oberflächen nach den CUA91-Regeln (Common User Access) zu erstellen. Die Einhaltung dieser Regeln war eine Anforderung, die mit Smalltalk umgesetzt werden konnte.
- Smalltalk bietet mit dem Model-View-Controler Application-Framework bereits eine umfangreiche Klassenbibliothek, die verwendet werden kann, um die Oberfläche zu realisieren.
- Smalltalk bietet einen Satz von Entwicklungswerkzeugen und wird interpretativ ausgeführt; dies erleichert die schnelle Konstruktion von Prototypen.

Der Entwicklungsprozeß

Das Projekt wurde so geplant, daß nach einer Analyse- und Entwurfsphase ein erster Prototyp erstellt werden sollte, der das Arbeiten mit SNE und die Oberfläche von SNE zeigen sollte. Anschließend sollte dieser erste Prototyp, der den Charakter eines Labormusters hatte, zu einem funktionalen Prototyp im engeren Sinn weiterentwickelt werden, der den Anwendern präsentiert werden kann.

Die Finanzierung des Projekts war bis zur Demonstration des zweiten Prototyps gesichert. Anschließend sollte entschieden werden, wie weiter zu verfahren ist. Der Aufwand für die Analyse betrug acht Personentage, der für den Entwurf 12 Personentage.

Auf der Basis des Entwurfs wurde der erste Prototyp in 14 Personentagen entwickelt. Als Vorbild für die Konstruktion der Oberfläche dienten existierende Nachrichten-Szenarien, die auf Papier vorlagen. Es wurde angestrebt, deren Aussehen auch mit dem Werkzeug zu erzielen.

In der Analyse und im Entwurf wurde die Booch-Notation für Klassen- und Objektdiagramme eingesetzt. Dies wurde ohne Werkzeugunterstützung getan, was nachträglich als Manko bewertet wird. Schritthaltend mit der Entwicklung der

konstruierten Prototypen wurden die erstellten "Booch-Templates" dokumentiert und die Klassen und Objektdiagramme aktualisiert. Eine Benutzungsdokumentation wurde nicht erstellt.

Die konstruierten Prototypen

Der erste Prototyp realisierte folgende Aspekte:

- Er zeigte, wie das geplante Werkzeug aussehen wird, indem das Layout der Fenster im Werkzeug und ihre Interdependenzen betrachtet werden konnten.
- Er demonstrierte, wie mit dem Werkzeug gearbeitet werden kann. Es konnten Bankfunktionen plaziert werden, und mit der Maus konnten Nachrichtenflüsse zwischen den Arbeitsplätzen eingerichtet werden.

Die gesamte fachliche Komponente und insbesondere der Generierungsmechanismus für die Entscheidungstabellen fehlten jedoch. Der erste Prototyp wurde von den Spezialisten des SWIFT-Systems, von den Entwicklern und von Mitarbeitern des Marketing bewertet. Dabei wurden Wünsche eingebracht, die die Präsentation des Editors betrafen. Weiterhin wurden auch Funktionen identifiziert, die der Editor noch nicht zeigte, die aber benätigt werden. Für die Bewertung des Prototyps gab es keine definierten Kriterien, sie geschah mehr oder weniger intuitiv.

Der erste Prototyp wurde anschließend iterativ zu einem funktionalen Prototyp in engeren Sinn weiterentwickelt. Der Aufwand dafür betrug 32 Personentage, der Prototyp besteht aus 60 - 100 Klassen. Dieser Prototyp zeigte nicht nur die Oberfläche und die Handhabung des Werkzeugs, sondern er generierte auch die gewünschten Assemblerteile. Weiterhin wurde bereits überprüft, ob die mit dem Werkzeug erstellten Nachrichten-Szenarien korrekt definiert sind. Ein kontextsensitives Hilfesystem wurde ansatzweise implementiert.

Der funktionale Prototyp wurde den potentiellen Anwendern gezeigt. Daraufhin wurde entschieden, auf dessen Basis die Produktentwicklung zu beginnen. Es werden dafür zwei Personenjahre geschätzt.

Die Produktentwicklung ist zur Zeit noch nicht abgeschlossen. Für die endgültige Entwicklung der Benutzungsoberfläche sind Grafiker und Spezialisten für Software-Ergonomie, aber auch Anwender am Projekt beteiligt worden. In den Entwicklungsprozeß werden fachliche und technische Reviews eingeplant; dies ist während der Prototypenentwicklung nicht geschehen.

In der Produktentwicklung arbeitet ein Kern des Teams mit, das die beiden Prototypen erstellt hat.

Erfahrungen der Entwickler

- Der Einsatz der Booch-Methode wurde als sehr hilfreich bewertet, jedoch sollte sie werkzeugunterstützt eingesetzt werden, damit die Systemdokumentation vereinfacht und vereinheitlicht wird.

- Die eingeplante Zeit für die Entwicklung der beiden Prototypen wurde deutlich unterschritten, weil einiges aus der Klassenbibliothek wiederverwendet werden konnte, und das interaktive System die Arbeit erleichtert. Der Laborprototyp wurde bereits nach 2/3 der eingeplanten Zeit fertig gestellt.
- Bei der iterativen Weiterentwicklung hätten häufiger Entwurfsphasen eingeschoben werden sollen. Design-€nderungen mußten deshalb während der Implementierungsphase durchgeführt werden. Die objektorientierte Konstruktion erleichterte jedoch dieses.
- Die Oberfläche von SNE wurde nach der MVC-Architektur realisiert, die Smalltalk/VPM in seiner Klassenbibliothek unterstützt. Da diese den Entwicklern zu Beginn unbekannt war, ergaben sich Probleme, die im Entwurf definierten Klassen für die Interaktion in das System zu integrieren. Es führte letzlich dazu, daß Teile des Entwurfs der Oberfläche nichtig wurden. Die bei der Analyse definierten Begriffe und Begriffshierachien im Bereich der zentralen Datenstrukturen des Werkzeugs konnten jedoch direkt in Klassenhierachien umgesetzt werden.
- Smalltalk/VPM ist als Werkzeug für diese Art von Anwendungen geeignet, da die Klassenbibliothek im Bereich der Benutzungsoberflächen die Konstruktion der Oberfläche wesentlich erleichtert hat.
- Smalltalk/VPM ist nicht für die Teamarbeit geeignet, da es keine Versionen- und Konfigurationsunterstützung gibt.
- Die mitgelieferte Dokumentation zu Smalltalk/VPM ist unzureichend. Es bedarf eines erheblichen Aufwands, um die Klassenbibliothek und die darin implementierten Architekturkonzepte zu verstehen.

Unsere Einschätzung

Dieses Projekt zeigt die Merkmale eines typischen Prototyping-Projekts, bei dem gleichzeitig neue Methoden (OOD) eingeführt und neue Werkzeuge (Smalltalk) eingesetzt werden. Prototyping wird evolutionär durchgeführt und es werden Prototypen für die unterschiedlichen Zwecke erstellt: zum einem konnte mit einem Prototyp die Handhabung der Anwendung gezeigt werden, zum anderen wurde ein funktionaler Prototyp erstellt, der den potentiellen Anwendern gezeigt werden konnte.

Das Application-Framework Smalltalk hat den Aufwand für die Entwicklung der Prototypen gering gehalten, weil sowohl im Bereich der Oberflächenkomponente als auch im Bereich der fachlichen Komponente existierende Klassen wiederverwendet oder durch Subklassenbildung angepaßt werden konnten.

Allerdings führt das Arbeiten mit einer Entwicklungsumgebung, wie sie von Smalltalk zur Verfügung gestellt wird, häufig dazu, daß zu früh mit der Implementierung begonnen wird, ohne daß ein tragfähiger Entwurf vorliegt. Dies mag an den sehr kurzen Entwicklungszyklen einer interpretativen Sprache liegen. Dies führt dann zu Strukturen, die nicht als Basis für die evolutionäre Entwicklung

zum Zielsystem verwendet werden kännen. Ein technisches Redesign ist dann unumgänglich.

Smalltalk als Prototyping-Werkzeug

Smalltalk kann unter verschiedenen Aspekten betrachtet werden Dazu zählen beispielsweise die Programmiersprache selbst oder die Entwicklungsumgebung. Nachfolgend soll Smalltalk unter dem Aspekt eines Application-Frameworks bewertet werden. Für ein tieferes Studium von Smalltalk verweisen wir auf die vorhandene Literatur.

Smalltalk enthält das MVC-Framework, um interaktive Anwendungen zu erstellten. Es bietet alle die Vorteile, die wir bereits in allgemeiner Form in Abschnitt 4.4.4 für Application-Frameworks beschrieben haben. MVC definiert eine Standard-Architektur für diese Anwendungen. Smalltalk stellt im Sinne eines Application-Frameworks Klassen zur Verfügung, die der Programmierer entweder direkt verwenden kann oder Unterklassen dazu bildet. Ist dieses Konstruktionsprinzip einmal verstanden, kännen Oberflächen architektonisch sehr sauber erstellt werden, die bereits eine Schnittstelle zur fachlichen Komponente der Anwendung definieren. Einen eigenen Interface-Builder, um die Oberflächenkomponente nach dem MVC-Konzept zu definieren, ist standardmäßig jedoch nicht in Smalltalk vorhanden – auf dem Software-Markt werden solche Werkzeuge allerdings angeboten (Digitalk PARTS, VisualWorks von ParcPlace oder VisualAge von IBM).

Smalltalk bietet als Programmierumgebung eine Vielzahl von Werkzeugen. Diese und die interpretative Ausführung führen zu kurzen Entwicklungszyklen, die auch für das Oberflächen-Prototyping von Vorteil sind.

Die von den Entwicklern geäußerten Probleme im Bereich Versionen- und Varianten-Management, wenn mehr als zwei Personen an der Entwicklung beteiligt sind, stimmen mit unseren Erfahrungen überein. Es existieren jedoch Systeme für Smalltalk wie z.B. ENVY/Developer, Team/V oder TeamBase, die Abhilfe schaffen oder wenigstens das Problem entschärfen.

5.4.2 Ein integriertes Werkzeug zur funktionalen Beschreibung technischer Systeme

Das untersuchte Projekt

Eine Firma, die mechatronische Systeme herstellt, hält regelmäßig Qualitätssicherungs-Sitzungen ab. Diese QS-Sitzungen dienen dazu, um Fehlerquellen in den Systemen aufzudecken und um die Abhängigkeiten zwischen verschiedenen Bausteinen zu untersuchen. An diesen QS-Sitzungen sind die Entwickler der Systeme und Mitarbeiter der Qualitätssicherung beteiligt. Die Ergebnisse einer solchen QS-Sitzung werden in speziell dafür vorgesehenen Formularen festgehalten. Anhand dieser Formulare wird das System bewertet, was zu Verbesserungsmaßnahmen führen kann. Zudem dienen die Formulare als Qualitätsnachweis für ausgesuchte Kunden.

Aus einer Forschungsabteilung kam nun die Idee, ein System zu entwickeln, mit dem mechatronische Systeme interaktiv am Bildschirm entwickelt werden kännen. Dieses System soll aus zwei Komponenten bestehen:

- Mit einem Editor sollen die einzelnen Komponenten eines Systems am Bildschirm angeordnet und der Signalfluß unter ihnen festgelegt werden kännen.
- Ein Modul soll zur Verfügung gestellt werden, mit dem die Fehler- und Signalflußanalyse des Systems durchgeführt werden kann.

Die Ergebnisse der Fehler- und Signalflußanalyse sollen benutzt werden, um eine QS-Sitzung des Systems vorzubereiten. Dies war unter Einsatz der herkämmlichen Methoden bis jetzt sehr zeitintensiv.

Die mit Hilfe des Systems modellierten Baugruppen oder Teilsysteme sollten so gestaltet sein, daß sie in späteren gräßeren Systemen jederzeit wiederverwendet werden kännen. Ein weiterer Gedanke ist, das erstellte Modell des Systems als Spezifikation für den späteren Entwicklungsprozeß zu verwenden.

Diese Idee wurde Abteilungen innerhalb der Firma unterbreitet, um interne Auftraggeber zu finden, die das Projekt finanzieren. Eine Abteilung aus dem Bereich Qualitätssicherung zeigte Interesse an der Idee. Sie erklärte sich bereit, das Projekt für die Dauer eines Jahres zu finanzieren. †ber die weitere Finanzierung des Projekts sollte dann in einem Jahresrhythmus entschieden werden.

Prototyping wurde aus folgenden Gründen für die gesamte Systementwicklung vorgesehen:

- Innerhalb der Entwicklungsmannschaft waren keine Erfahrungen auf dem Gebiet der Qualitätssicherung (speziell Fehler- und Signalflußanalyse) von mechatronische Schaltungen vorhanden.
- Zu Beginn des Projekts konnten aus den entsprechenden Abteilungen der Qualitätssicherung keine Mitarbeiter integriert werden.
- Es konnte keine detaillierte Spezifikation des gewünschten Systems erstellt werden.

Benutzte Werkzeuge

Das zu entwickelnde System sollte nach Ansicht der Forschungsabteilung über eine einheitliche grafische Benutzungsoberfläche verfügen, damit alle Werkzeuge flexibel miteinander kombiniert werden kännen. Zudem war man sich in der Forschungsabteilung einig, daß sich die Oberflächen der Werkzeuge nicht nur aus grundlegenden Oberflächenelementen (Knäpfe, Listen, etc.) aufbauen lassen. An die verwendete Entwicklungsumgebung wurden deshalb zwei Anforderungen gestellt:

- Die Oberflächen der Werkzeuge sollten schnell und einfach gestaltet werden kännen.

- Die Entwicklungsumgebung sollte um grafische Oberflächenelemente erweitert werden kännen, die selbst erstellt werden mußten.

Der Einsatz von Smalltalk-80 auf UNIX-Workstations wird wie folgt begründet:

- Das System sollte objektorientiert entwickelt werden.
- Smalltalk-80 ist auf verschiedenen Hardware- und Betriebssystemplattformen vorhanden.
- Smalltalk-80 bietet mit dem MVC Application-Framework eine gute Architektur für die Oberflächenkomponenten.
- Smalltalk-80 besitzt bereits eine sehr umfangreiche Klassenbibliothek. Zudem kännen für viele Anwendungsbereiche Klassenbibliotheken erworben werden.
- Der Quell-Code für sämtliche Klassen ist vorhanden und kann gegebenenfalls angepaßt werden.
- Die Entwickler hatten bereits Erfahrung mit Smalltalk-80.

Da Smalltalk-80 die Teamarbeit nicht unterstützt, wurde das Werkzeug Application-Organizer und eine darauf aufbauende eigene Versionenverwaltung eingesetzt. Dazu wurde der UNIX "make-Mechanismus" und eigene entwickelte Werkzeuge verwendet.

Der Entwicklungsprozeß

Mit der Implementierung des ersten Prototyps wurde im Dezember 1990 begonnen. Weder die Auftraggeber noch die Benutzer haben diesen Prototyp evaluiert. Aus ihm wurde eine textuelle Spezifikation des zukünftigen Systems erstellt, die dem Auftraggeber vorgelegt wurde. Die Entwickler sammelten bei der Konstruktion des Prototyps Vorstellungen und Ideen, wie eine optimale Werkzeugunterstützung für das angestrebte System aussehen kännte. Zudem erwarb man Erfahrungen, wie das System mit Smalltalk-80 software-technisch realisiert werden konnte.

Aufbauend auf diesem Prototyp wurde ein weiterer Prototyp mit dem Ziel entwickelt, ihn dem Auftraggeber und den Benutzers zur Evaluation zu präsentieren. Begonnen wurde mit ihm Mitte des Jahres 1991. Die Präsentation erfolgte auf einer firmeninternen Vorführung im Frühjahr 1992.

Die Demonstration führte zu sehr unterschiedlichen Meinungen bei den Benutzern (von "sehr gut" bis "was soll das"). Dies war auf den bis dahin geringen Einsatz der Informationstechnik in den Arbeitsbereichen der Qualitätssicherung zurückzuführen. Die durch die Präsentation entstandene Diskussion ergab die Anforderung, das System besser in die im Unternehmen bestehende Vorgehensweise einzubetten. Dies bedeutet, daß die Ergebnisse der Fehler- und Signalflußanalyse eines Systems in der für das Unternehmen bestehenden Formularform ausgedruckt werden sollen. Zudem ergab sich die Forderung, daß das Werkzeug Simulationsmodelle, die bereits vorhanden waren, verarbeiten kann. An der Gestaltung der Oberfläche gab es wenig Kritik. Die informationstechnische Verarbeitung der Ergebnisformulare einer Fehler- und Signalflußanalyse läste bei

den Benutzern erstmals eine intensive Diskussion über die Semantik dieser Formulare aus. Dies bezog sich auf den Zusammenhang der einzelnen Daten, die auf dem Formular erfaßt wurden. Erstaunlich ist dabei, daß die Mitarbeiter schon lange Zeit mit den Formularen in Papierform gearbeitet haben.

Der dritte Prototyp wurde von den Entwicklern mit dem Ziel erstellt, ihn als Pilotsystem einzusetzen. Es konnten jedoch keine Benutzer für den Piloteinsatz des Systems gefunden werden. Dies wurde gräßenteils mit Zeitmangel begründet.

Um die Flexibilität und die Erweiterbarkeit des Systems zu verbessern, entstand durch ein tiefgreifendes fachliches und technisches Redesign ein weiterer vierter Prototyp. Für ihn gibt es innerhalb der Firma vier Pilotinterssenten. Eine erste Installation bei einem Benutzer war für Ende 1992 geplant.

Bei der Entwicklung des Systems war von Anfang an eine externe Firma integriert. Die erste Zusammenarbeit fand auf der Basis von konkreten Teilaufträgen statt. Mitarbeiter des externen Partners nahmen auch am technischen und fachlichen Redesign des Systems teil, weil in der Firma selbst das System nicht vermarktet und gewartet werden konnte. Dieses sollte der externe Partner tun. Der potentielle Kundenstamm für das Produkt ist derzeit aber nur firmenintern. An einer Vermarktung des Produkts außerhalb der Firma wird derzeit noch nicht gedacht. Der Aufwand, der bis jetzt von der externen Firma innerhalb des Projekts geleistet wurde, beträgt ca. vier Personenmonate.

Um die Oberfläche der Werkzeuge zu gestalten, wurden keine Designer oder Grafiker eingesetzt. Die Entwickler entwarfen sie aufgrund der Erfahrungen aus früheren Tätigkeiten. Weiterhin wurde die Gestaltung der Oberfläche von anderen Systemen wie dem Macintosh beeinflußt. Speziell die Oberfläche des ersten Prototyps lehnte sich stark an ein regeltechnische Simulationswerkzeug an, das bereits innerhalb der Firma entwickelt wurde.

Derzeit sind nur Teile des Systems dokumentiert. Es existiert eine Entwurfsspezifikation für den zweiten Prototyp sowie für Teile des dritten Prototyps. Für den vierten Prototyp ist die Spezifikation in Arbeit. Entwurfsdokumente und ein Benutzungshandbuch sind noch nicht vorhanden. Ausführlich wurden nur die Aufträge an die externe Firma beschrieben. Allerdings ist sich die Entwicklungsmannschaft bewußt, daß eine bessere Dokumentation des Systems erforderlich ist.

Die konstruierten Prototypen

Prototyp I: Dieser Prototyp wurde in ca. 1/2 jähriger Arbeit fertiggestellt und kann als horizontales Labormuster bezeichnet werden. Er wurde von einem Entwickler ohne Einbeziehung der Benutzer erstellt. Er besteht funktional aus einem Werkzeug, mit dem interaktiv mechatronische Systeme modelliert werden kännen. Die Klassenbibliotheken NEDT und VICK wurden dabei verwendet und auf die speziellen Bedürfnisse für das Projekt angepaßt. Der Aufwand betrug ca. vier Personenmonate. Der Prototyp besteht aus ca. 10-15 Klassen.

Prototyp II: Aus der durch den ersten Prototyp gewonnenen Spezifikation wurde von zwei Entwicklern innerhalb eines 1/2 Jahres ein zweiter Prototyp erstellt. Der erste Prototyp diente dabei als "Steinbruch". Am Erscheinungsbild und an der Handhabung des Editors änderte sich im Bezug auf den ersten Prototypen nichts. Der Aufwand betrug ca. sieben Personenmonate.

Prototyp III: Er wurde evolutionär aus dem zweiten Prototyp entwickelt und deckt die von den Benutzern geforderte Funktionalität, Formulare alter Art zu erstellen und bestehende Simulationsmodelle einzulesen, ab. Der Prototyp besteht aus ca. 50-60 Klassen.

Prototyp IV: Nachdem der dritte Prototyp fertig gestellt war, stellten die Entwickler fest, daß dieser so realisiert war, daß die fachliche Komponente und die Oberfläche sehr stark gekoppelt waren. Um das System flexibler gestalten zu kännen, wurde von den Entwicklern ein technische Redesign vorgenommen, was zu einer strikten Trennung der beiden Teile führte. Dies war nätig, da verschiedene Sichten auf dieselben Daten benätigt wurden. Im Bezug auf den Qualitätssicherungsprozeß ergaben sich keine €nderungen im System. Der Aufwand für den dritten und vierten Prototyp wird auf ca. ein Personenjahr geschätzt.

Erfahrungen der Entwickler

Insgesamt waren die Entwickler mit der Durchführung des Projektes zufrieden. Sie empfanden es allerdings als nachteilig, daß die Benutzer erst sehr spät in den Entwicklungsprozeß eingezogen wurden. Als Grund dafür geben sie an, daß die Benutzer nicht überzeugt werden konnten, daß der damit verbundene Aufwand letzlich rentabel ist.

Der Einsatz von Smalltalk-80 bei der Entwicklung des Systems wurde insgesamt positiv bewertet. Die schlechte Unterstützung der Teamarbeit und die Schwierigkeiten, bestehende Systeme in Smalltalk-80 zu integrieren, wurden als Kritikpunkte genannt.

Bei einer erneuten Durchführung des Projekts würden die Entwickler auf eine bessere Dokumentation des Entwurfs achten. Hier wäre ihrer Meinung nach eine bessere Unterstützung des Smalltalk-Systems hilfreich. So wurde es als negativ empfunden, daß Entwurfsentscheidungen nicht geeignet abgelegt werden kännen. Sie kännen nach Meinung der Entwickler nicht an eine Klasse geknüpft werden. Zudem besteht nicht die Mäglichkeit, Kategorien von Klassen zu dokumentieren.

Unsere Einschätzung

Der erste Prototyp diente dazu, auf der Seite der Entwickler ein erstes Bild des späteren Systems zu schaffen und die verwendete Entwicklungsumgebung kennenzulernen. Dieser sollte deshalb innerhalb in einer kurzen Entwicklungszeit erstellt werden kännen, da er später als "Steinbruch" verwendet wird.

Dieses Projekt zeigt dieselben zentralen Probleme beim Einsatz von Smalltalk, die schon im ersten beschriebenen Smalltalk-Projekt von den Entwicklern geäußert

wurden: Mangelnde Dokumentation, die das Einarbeiten in das Framework und in die Klassenbibliothek erschweren und eine fehlende Unterstützung beim team-orientierten Entwickeln.

Ebenfalls typisch für das evolutionäre Arbeiten mit einem Framework ist unserer Meinung nach, daß sich das Design der Anwendung im Rahmen der vorgegebenen Architektur weiterentwickelt. Dabei muß jedoch darauf geachtet werden, daß die Qualität der Architektur nicht abnimmt. Notwendige technische Redesigns sind – wie das Projekt zeigt – dazu erforderlich.

5.4.3 Ein Swaps-Manager

Das untersuchte Projekt

Die Swaps-Abteilung einer Bank handelt mit derivativen Zinsprodukten. Abschlüsse werden dabei telefonisch getätigt.

Zu Beginn des Projektes setzten die Swaps-Händler ein PC-basiertes System ein, mit dem nach Abschluß eines Handels gewisse Aspekte nachgerechnet werden kännen. Die großen Schwächen dieses Systems sind die finanzmathematische Unvollständigkeit und die mangelhafte Benutzungsoberfläche. Diese ließ es nicht zu, daß ein Händler mit dem System Bewertungen durchführt, während er telefonisch einen Handel abschließt.

Aufgrund dieser offensichtlichen Mängel wurden zwei Projekte gestartet. Das Ziel des ersten Projektes war, innerhalb relativ kurzer Zeit WASP, ein finanzmathematisch vollständiges Nachfolgeprodukt des existierenden Systems, zu entwickeln. Das Ziel des zweiten Projektes war, mittelfristig den SWAPSMANAGER, ein finanzmathematisch vollständiges Produkt, das auch während des telefonischen Handels eingesetzt werden kann, zu entwickeln.

In dieser Fallstudie gehen wir nur noch auf die Entwicklung des SWAPSMANAGERs ein. An diesem Projekt waren die Swaps-Abteilung und die DV-Forschungsabteilung der Bank beteiligt. Es kam aufgrund informeller Kontakte zustande. Dabei wurden zwei Ziele verfolgt:

- Die Swaps-Händler erwarten als Resultat einen ergonomisch und funktional wesentlich verbesserten Arbeitsplatz, der es ihnen erlaubt, während des telefonischen Handels Swaps und Portfolios von Swaps zu definieren und unter verschiedenen Annahmen zu bewerten.
- Die Forschungsabteilung will mit dem Projekt zeigen, daß die intern entwickelte Technologie einen gewissen Reifegrad erreicht hat.

Vor Beginn des Projekts mußten keine Verträge abgeschlossen werden, da die Forschungsabteilung die Entwicklungskosten trägt. Es wurden einzig zwei "Sollbruchstellen" definiert, nach denen jede der beteiligten Parteien die Zusammenarbeit einstellen konnte. Sollbruchstelle eins war die Fertigstellung des ersten Prototyps. Mit ihm sollte überprüft werden, ob es überhaupt mäglich ist, eine Benutzungsoberfläche zu entwickeln, die man während des telefonischen Handels

verwenden kann. Sollbruchstelle zwei war das Fertigstellen der Spezifikation des Pilotsystems, das als explorativer Prototyp dienen sollte.

Die Entwicklungs- und Zielsysteme sind UNIX-Workstations. Als Entwicklungswerkzeug wurde das Application-Framework ET++ eingesetzt.

Der Entwicklungsprozeß

Phase I: Studium des Anwendungsgebiets und mäglicher technischer Läsungen

In der ersten Phase ging es für die Entwickler darum, das Anwendungsgebiet kennenzulernen und in einem iterativen Prozeß zusammen mit den Anwendern einen Prototyp zu erstellen. Dieser sollte bei beidseitiger Zufriedenheit als Spezifikation und Basis für ein Pilotsystem dienen.

Diese Phase dauerte sieben Monate. In dieser Zeit war ein Entwickler vollständig mit der Implementierung des Prototyps beschäftigt. An der Entwicklung neuer Ideen für die Benutzungsoberfläche haben zwei weitere Entwickler mitgewirkt. Dies aber nur in einem Bruchteil ihrer Arbeitszeit.

Folgende Aktivitäten wurden in Phase I durchgeführt:

- Erarbeiten der bankfachlichen und mathematischen Grundlagen durch die Entwickler in enger Zusammenarbeit mit den Anwendern. Die Entwickler mußten sich so weit in die Materie einarbeiten, daß sie die Arbeitsabläufe verstanden und in der Lage waren, realistische Prototypen zu bauen. Das Einarbeiten in die finanzmathematischen Grundlagen war selbst für einen Wirtschaftsinformatiker sehr aufwendig.
- Entwickeln eines ersten Prototyps mit moderner Benutzungsoberfläche, mit dem schnell und einfach Portfolios verwaltet, definiert und bewertet werden konnten. Diese Aktivität war ein zyklischer Prozeß, in dem die Entwickler Vorschläge erarbeiteten und sie dann zusammen mit den Anwendern evaluierten. Um eine enge Zusammenarbeit sicherzustellen, haben die Entwickler zeitweise bei den Swaps-Händlern programmiert. Aufgrund der technischen Komplexität dauerten die Implementierungs-Evaluations-Zyklen von wenigen Tagen bis zu einem Monat.
- Weiterentwickeln dieses Prototyps, bis er von beiden Seiten als Spezifikation für das Pilotsystem akzeptiert wurde.

Der Schwerpunkt bei der Entwicklung des Prototyps lag eindeutig auf der Benutzungsoberfläche. Die Funktionalität wurde nur soweit implementiert, daß die Standardfälle korrekt unterstützt wurden. Der Anschluß an die bestehenden Daten und die Sonderfälle in der Abwicklung wurden nicht berücksichtigt.

Der Entwickler des ersten Prototyps verließ die Forschungsabteilung, nachdem die Phase I abgeschlossen war. Es dauerte deshalb ca. sechs Monate, bis Phase II mit einem neuen Entwickler beginnen konnte.

Phase II: Weiterentwicklung des Prototyps zu einem Pilotsystem

Das Ziel der zweiten Phase war, aus dem Prototyp ein einsatzfähiges Pilotsystem zu entwickeln. Die Entwickler hatten weiterhin das Ziel, aufgrund der Erfahrung aus Phase I und II allgemein wiederverwendbare Bausteine für Händler-Anwendungen zu entwickeln.

Phase II dauerte ca. ein Jahr. In dieser Zeit haben zwei Entwickler zu 100% am Pilotsystem gearbeitet.

Folgende Aktivitäten wurden in Phase II durchgeführt:

- Erarbeiten der exakten bankfachlichen und mathematischen Grundlagen durch die Entwickler. In dieser Phase haben sich die Entwickler in enger Zusammenarbeit mit dem zuständigen Finanzanalytiker und durch extensive Lektüre der Fachliteratur ein vollständiges Verständnis der Materie erarbeitet.
- Entwickeln einer Datenbankschnittstelle, die es der objektorientiert konstruierten Anwendung transparent ermäglicht, auf Informationen in einer bestehenden und einer zu entwickelnden relationalen Datenbank zuzugreifen.
- Integration von Bärsendaten in Realzeit.
- Implementierung und Test der Rechnungskomponente.
- Inkrementelle Weiterentwicklung der Benutzungsoberfläche.

In Phase II wurden die Anwender nicht mehr zur Evaluation des Systems beigezogen. Der Entwicklungsprozeß war nur noch dort zyklisch, wo man verschiedene technische Läsungen ausprobierte und wo Generalisierungs-mäglichkeiten gefunden wurden.

Phase III: Weiterentwicklung des Pilotsystems zu einem Produkt

Diese Phase hat noch nicht begonnen.

Erfahrungen der Entwickler

- Der Aufwand, um das benätigte Fachwissen zu erarbeiten, war wesentlich gräßer, als zuerst angenommen. Dieser Prozeß konnte nicht sinnvoll mit Prototyping unterstützt werden. Prototyping war aber in Phase I und II wichtig, um das Fachwissen zu †berprüfen.
- Es war schwer, die Anwender für dieses Projekt zu interessieren, da es lange dauerte, bis der erste Prototyp erstellt war, mit dem man zeigen konnte, daß die geplante Art von Benutzungsoberfläche ihre Bedürfnisse wirklich befriedigt. Dies war aufgrund der Neuartigkeit und Komplexität der Benutzungsoberfläche nicht zu vermeiden.
- Der erste Prototyp wurde inkrementell und evolutionär zum Pilotsystem weiterentwickelt. Während dieser Zeit war der Prototyp zeitweise explorativ, experimentell und evolutionär. Ohne einen evolutionären Prototyping-Ansatz wäre die Entwicklung eines solchen Produktes kaum mäglich gewesen.
- Der Einsatz des ET++ Application-Frameworks wird als sehr positiv bewertet. Es erlaubt durch seine wiederverwendbaren Bausteine, relativ schnell

Standardbenutzungssoberflächen zu bauen. Komplexe neuartige Komponenten konnten, aufbauend auf den Framework- und Visualisierungsklassen, in relativ kurzer Zeit implementiert werden. Der wichtigste Aspekt war aber, daß die in weiten Teilen vorgegebene Anwendungsarchitektur ermäglichte, den Prototyp evolutionär weiterzuentwickeln, ohne daß regelmäßig gräßere Redesigns nätig waren.

Unsere Einschätzung

Das Swaps-Manager Projekt ist ein gutes Beispiel, für ein Entwicklungsprojekt, im Rahmen dessen verschiedene Arten von Prototyping auftraten. Bemerkenswert sind dabei die folgenden Aspekte:

- Zu Beginn diente Prototyping nicht dazu, den Benutzern Wissen zu vermitteln, sondern dazu, das Verständnis des Anwendungsbereich durch die Entwickler zu validieren.
- Die Entwicklung des Pilotsystems war ein evolutionärer Prozeß, in dem der Prototyp zu einer praktisch einsetzbaren Anwendung ausgebaut wurde. Es darf aber nicht übersehen werden, daß dies nur durch regelmäßiges Redesign mäglich war. Application-Frameworks unterstützen dieses Vorgehen, weil sie eine Standardarchitektur vorgeben. Der Aufwand für die Redesigns darf aber nicht unterschätzt werden.
- Wir glauben nicht, daß die Anwender bereit gewesen wären, die effektiven Kosten für den gewählten Ansatz zu tragen. Wenn aber nicht eine gewisse Anzahl Anwender bereit dazu ist, dann wird es nie mäglich sein, für spezifische Anwendungsgebiete, wie für Händleranwendungen, wiederverwendbare Frameworks zu erstellen.

5.4.4 Auswertung der Werkzeuggruppe

Application-Frameworks werden meistens eingesetzt, um interaktive Anwendungen zu erstellen, deren Benutzungsoberflächen nicht einfach aus Standardoberflächen-elementen (Knäpfe, Textfenster, usw.) zusammengesetzt werden kännen. So enthalten alle drei Anwendungen einen interaktiven grafischen Editor, der jeweils ganz speziellen Anforderungen genügen muß: In der ersten Fallstudie ist dies der Swift-Nachrichten Editor, in der folgenden ein Editor zum Gestalten von mechatronischen Systemen und in der dritten Fallstudie ist es ein Editor, um Portfolios definieren, verwalten und bewerten zu kännen.

Prototypen, die mit Hilfe eines Application-Frameworks erstellt werden, unterliegen meistens einem evolutionären Zyklus. Dies liegt sicherlich an dem großen Anteil an "Handarbeit", der bei der Erstellung eines Prototyps mit Hilfe eines Application-Frameworks notwendig ist.

Application-Frameworks kännen aber auch verwendet werden, wenn ein Demonstrationsprototyp konstruiert werden muß, der neben der Oberfläche auch Teile der Funktionalität zeigen soll. Dieser kann bei Kenntnis des Frameworks mit vertretbarem Aufwand erstellt werden. Der resultierende Prototyp-Code ist in

diesem Fall jedoch nicht als Basis für einen evolutionären Entwicklungsprozeß geeignet (quick and dirty) und muß weggeworfen werden. Er kann letztlich nur als "Steinbruch" für weitere Prototypen dienen

Herausragendes Merkmal von Application-Frameworks ist die gute Wiederverwendbarkeit von Bausteinen und Architekturprinzipien. Dies wird sicherlich durch die Verwendung von objektorientierten Programmiertechniken begünstigt, was die geschilderten Fallstudien bestätigt haben.

Application-Frameworks kännen besonders effizient für das Oberflächen-Prototyping eingesetzt werden, wenn es einen darauf abgestimmten Interface-Builder gibt. Fehlt der Interface-Builder oder reicht sein Leistungsumfang nicht aus, so entsteht ein entsprechend häherer Programmieraufwand für die Oberflächenkomponenten.

Generell gilt, daß ein nicht unerheblicher Lernaufwand notwendig ist, um ein Application-Framework im intendierten Sinn verwenden zu kännen. Dieser Aufwand darf nicht unterschätzt und muß eingeplant werden.

Kapitel 6

Analyse und Bewertung

In diesem abschließenden Kapitel wollen wir resümieren, was aus unserer Sicht Erfahrungen, Konzepte und Entwicklungslinien beim Prototyping von Oberflächen sind. Dabei beziehen wir uns zunächst auf die Gespräche und Analysen, die unmittelbar zu dieser Studie geführt haben. Darüber hinaus haben wir natürlich unseren jeweiligen beruflichen Hintergrund einfließen lassen. Alle Mitglieder des Autorenteams beschäftigen sich seit längerer Zeit intensiv in verschiedenen Bereichen mit Fragen des Prototyping, der benutzerorientierten Software-entwicklung und mit der Konstruktion und dem Einsatz von Werkzeugen zum Oberflächen-Prototyping. Wir sprechen aber nicht nur Fragen an, die sich auf Oberflächen-Prototyping im engeren Sinnen beziehen. In diesem Resümee wollen wir insgesamt aufzeigen, wo Prototyping heute steht, d.h. wie Prototyping-Projekte heute gestaltet werden und welche Erfahrungen sich dabei verallgemeinern lassen. Deshalb sprechen wir auch Aspekte an, die in den vorausgegangenen Teilen dieser Studie nicht behandelt wurden, die aber u.E. das Bild abrunden.

6.1 Prototyping als Prozeß

Prototyping darf und kann nicht als ein rein technisches Verfahren verstanden werden, das sich beliebig in den Software-Entwicklungsprozeß einbauen läßt und dabei so wenig "Seiteneffekte" zeigt, wie die Auswahl eines Texteditors oder eines Darstellungsmittels für Programmentwürfe. Prototyping ist nicht nur eng mit einer übergreifenden Entwicklungsstrategie verknüpft, es muß auch immer im Zusammenhang mit einer bestimmten Sichtweise der Softwareentwicklung gesehen werden. Dies bedeutet in der Praxis, daß die Art der Projektorganisation, die Wahl eines Leitbildes für die Softwareentwicklung und die Vorgehensweise mit darüber entscheiden, ob Prototyping als Verfahren nutzbringend und zielführend ist. Im weiteren werden wir dies erläutern.

6.1.1 Leitbild und neue Sichtweise

Das Leitbild bei der Softwareentwicklung prägt den Prototyping-Prozeß entscheidend. Daher ist in jedem Projekt zum frühest mäglichen Zeitpunkt zu klären, welche (Wert-) Vorstellungen und damit leitenden Ideen mit dem Entwicklungsprozeß verbunden sind. Grundsätzliche Orientierungen sind auf der einen Seite die Idee der mäglichst vollständigen Ablaufsteuerung und Automatisierung, d.h. das Bild der Fließbandarbeit, und auf der anderen Seite die Idee von Software als unterstützendem Hilfsmittel für qualifizierte menschliche Arbeit, d.h. das Bild vom gut ausgestatteten Arbeitsplatz. Jedes Softwareprojekt wird sich seinen

Platz in diesem Spektrum suchen müssen. Dieser Platz bestimmt aber die Mäglichkeiten, die anwendungs- und benutzungsorientiertes Prototyping bietet: Eine rigide Ablaufsteuerung und Prozeßautomatisierung als Leitbild wird selten zu einer kooperativen und offenen Arbeit mit Prototypen zwischen Anwendern, Benutzern und Entwicklern führen, da nur wenige Anwender und Benutzer eine fließbandartige Tätigkeit als ihre Wunschvorstellung bei der Softwareentwicklung haben werden. Entwickler, die eine solche Art von Software anstreben, haben sicherlich ein positives Bild von Anwendern.

Neben dem Leitbild spielt die Sichtweise auf den Entwicklungsprozeß eine wichtige Rolle. Vielfach dominiert die sog. Produktsicht, d.h. Softwareentwicklung wird als ein Produktionsprozeß verstanden, der ähnlich der industriellen Herstellung von Konsumgütern in einem festgelegten und mäglichst arbeitsteiligen Produktionsprozeß zu einem vorab definierten Produkt führt. Entsprechend sind viele Entwicklerorganisationen in Abteilungen wie Kundenbetreuung, Entwicklung, Organisation, Vertrieb, Infrastruktur etc. aufgeteilt. In den letzten Jahren ist deutlich geworden, daß Software so kaum anwendungsorientiert entwickelt werden kann. An die Stelle der Produktsicht tritt vermehrt eine Projekt- oder Prozeßsicht, die in der Softwareentwicklung vor allem als Lern- und Kommunikationsprozeß aller beteiligten Gruppen gesehen wird. Die Folgen für das Prototyping sind offensichtlich. Während die Produktsicht Prototyping vorrangig als Instrument der Anforderungsanalyse einsetzt, versteht die Prozeßsicht Prototyping als grundlegendes Mittel zur Färderung von Lernprozessen und als Grundlage der Kommunikation und Kooperation im gesamten Entwicklungsprozeß. Dies hat seinerseits Auswirkungen auf die Organisation, da sich die klassische Abteilungsstruktur als sperrig für solche anwendungs- und prozeßorientierten Projekte mit integrierten Teams erweist. Wir werden auch darauf im weiteren noch eingehen.

Wenn Softwareentwicklung anwendungs- und benutzerorientiert gesehen wird, dann muß Prototyping als Bestandteil einer evolutionären Vorgehensweise Ausdruck einer neuen Projektkultur sein. Eine Projektkultur ist dann entstanden, wenn diese Vorgehensweise der Normalfall ist und nicht speziell betont werden muß. Dazu müssen traditionelle Vorgehensmodelle ersetzt werden, aber die Entwicklerorganisation muß das neue Konzept auch ihren Kunden gegenüber verdeutlichen. Dies hat Auswirkungen bis in die Vertragsgestaltung.

6.1.2 Vorgehensmodell

Werkzeuge kännen nur dort einen Nutzen erbringen, wo sie in einem stimmigen Gesamtkonzept eingesetzt werden. Das heißt zunächst und vor allem, daß die Vorgehensweise klar und auf den Einsatz von Prototyping ausgerichtet ist. Weiterhin ist wichtig, daß die fachliche Zielsetzung eines Softwareprojektes geklärt ist, oder, wenn dies beim Projektstart noch nicht der Fall ist, daß diese fachliche Klärung vorrangig im Projekt angestrebt wird. In solchen Fällen kommt dem Werkzeugeinsatz, auch für das Oberflächen-Prototyping, nur nachrangige Bedeutung zu.

Klarheit des konzeptionellen Rahmens ist vielfach dort gegeben, wo die Anwendung von der fachlichen Seite her gut bekannt ist und wo Benutzer und Anwender bereits

selbstverständlich mit DV-Unterstützung arbeiten. In solchen "Standardanwendungen" finden wir auch viele bewährte Werkzeuge, die es erlauben, in relativ kurzer Zeit durch Komposition vorhandener Bausteine Teile von Anwendungen zu bauen. Bei den untersuchten Projekten kännen 4. Generationssysteme als Beispiel genommen werden.

Die zeitlich-logische Abfolge beim Bau von Oberflächenprototypen ist von Bedeutung. Obwohl vielfach noch propagiert wird, macht es u.E. wenig Sinn, zuerst die Benutzungsoberfläche zu entwerfen und dann daraus die fachliche Komponente der Anwendung abzuleiten. Dies führt in der Regel zu Softwarearchitekturen, die keine saubere Trennung von Darstellung und Handhabung an der Oberfläche sowie fachlicher Funktionalität haben. Zudem wird bei diesem Vorgehen die Diskussion über die fachliche Substanz des Anwendungssystems vollkommen vernachlässigt. Dies führt z.B. in objektorientierten Systemen oft dazu, daß die fachliche Komponente software-technisch explizit als reine "Datenobjekte" abgebildet wird, wobei dann große Teile der fachlichen Dynamik implizit in der Oberflächenkomponente realisiert werden. Architekturen dieser Art sind schwer verständlich und kaum weiterentwickelbar. Ein zusätzliches Problem stellt die analysierte "Form-Inhalt"-Problematik dar. So verlockend es für Entwickler sein mag, zunächst ein Oberflächen-Layout mit den Benutzern abzustimmen und dieses dann schrittweise mit fachlicher Funktionalität anzureichern, so wenig tragfähig ist diese Vorgehensweise, wenn die fachlichen Inhalte der zukünftigen Anwendung den Beteiligten prinzipiell noch nicht klar sind. Wir sehen allerdings, daß es am Markt viele Werkzeuge gibt, die dieses Vorgehen nahelegen. Eine der wenigen positiven Ausnahmen stellen eng gekoppelte 4G-Systeme wie 4th-Dimension dar. Bei diesen 4G-Systemen muß zunächst (graphisch) ein Datenmodell entworfen werden, ehe dazu passende Oberflächenkomponenten erzeugt werden kännen.

Dies soll allerdings nicht den Umkehrschluß nahelegen, daß "gute" Anwendungen dadurch entstehen, zuerst ein vollständiges fachliches Modell zu entwerfen und dann davon eine Oberfläche abzuleiten. Gute Oberflächen zeichnen sich unseres Erachtens nach nicht dadurch aus, dem Benutzer ein 1:1-Abbildung des fachlichen Modells zu präsentieren, sondern ihm die Informationen eines solchen Modells geeignet darzustellen und sie damit bearbeitbar zu machen. So ist es z.B. in Bankanwendungen üblich, dem Benutzer (Bankmitarbeiter) mit Hilfe eines EDV-Systems einen "Gesamtüberblick" über die Vermägensverhältnisse eines Kunden zu geben, obwohl sich ein "Gesamtüberblick" normalerweise nicht als Objekt in der fachlichen Komponente der Anwendung befindet. Um somit Anwendungen zu entwickeln, deren fachlicher Kern stimmig ist und deren Handhabung und Präsentation der Oberfläche den Wünschen der Benutzer entspricht, ist eine Benutzerbeteiligung und ein insgesamt evolutionäres Vorgehen unumgänglich.

Prototyping als Vorgehensweise muß selbst zum Gegenstand der Diskussion zwischen Entwicklern und den anderen Gruppen werden. Denn nur wenn der Stellenwert eines Oberflächenprototyps klar ist und wenn die einzelnen Gruppen um ihre Einflußmäglichkeiten im Prototyping-Prozeß wissen, kann das Potential dieser Vorgehensweise voll ausgeschäpft werden. Erfahrungen zeigen, daß der implizite oder "verdeckte" Einsatz von Prototypen eher kontraproduktiv ist. Damit meinen wir

eine Situation, in der die Entwickler den Benutzern nicht klarmachen, daß sie einen Prototyp bewerten sollen und daher die Benutzer den Eindruck haben, mit einem unausgereiften Zielsystem konfrontiert zu sein.

6.1.3 Anwendungsorientierung und Benutzerbeteiligung

Wir haben Prototyping als Prozeß in eine evolutionäre Strategie eingeordnet. Diese Strategie ist in ihrer Zielsetzung primär anwendungsorientiert, d.h. die Qualität des zu entwickelnden Softwareprodukts wird vorrangig an der Benutzbarkeit im Anwendungsbereich ausgerichtet. Aber Prototyping bedeutet heute nicht nur Anwendungsorientierung, sondern auch Benutzerbeteiligung. Gerade beim Oberflächen-Prototyping ist diese Ausrichtung für den Projekterfolg entscheidend.

Eine Konsequenz der Anwendungsorientierung ist, daß sich die Entwickler selbst intensiv mit den Aufgaben und Tätigkeiten im Anwendungsbereich beschäftigen und die dahinstehenden Konzepte verstehen müssen.

Benutzerbeteiligung bedeutet, daß diejenigen einbezogen werden müssen, die später zumindest potentiell das Softwaresystem benutzen, denn nur sie kännen letztlich die Qualität der Oberfläche und die fachliche Stimmigkeit beurteilen. Die Integration der Benutzer sollte daher nicht nur zur Gestaltung und Bewertung eines Oberflächenprototyps erfolgen.

Um "gutes" Prototyping zu betreiben, ist es wichtig, die Benutzer so früh wie mäglich, aber dann auch für den gesamten Projektverlauf zu beteiligen. Dies hat, wie oben angedeutet, Auswirkungen auf die gesamte Durchführung eines Projekts. Neben der Diskussion über Prototypen müssen generell neue Wege und Mittel eingesetzt werden, um ein Projekt für alle beteiligten Gruppen verständlich durchzuführen.

Kontinuierliche Benutzerbeteiligung führt nicht als solche schon zum Erfolg. Die Position oder Rolle, die die Benutzer einnehmen, ist entscheidend. Zunächst gilt, daß die Benutzer aktiv und gleichberechtigt mit einbezogen werden müssen. Negativ erweist sich, wenn Benutzer nur als Informationslieferanten und als "Versuchskaninchen" eingesetzt werden, d.h. wenn sie den Eindruck erhalten, daß Prototyping nur dazu dient, aus ihnen Fachwissen "herauszuziehen", ohne daß sie an konzeptionellen Entscheidungen über die Gestaltung der Prototypen beteiligt sind. Diese Rolle der Benutzer als Informationslieferanten zeigt sich oft daran, daß sie in entscheidenden Sitzungen durch sog. Benutzervertreter aus dem mittleren Management ersetzt werden. Von dort bis hin zu gleichberechtigten Partnern in einem integrierten Team ist gedanklich und organisatorisch ein weiter Weg.

Die Einbeziehung der Benutzer in den Entwicklungsprozeß sollte nicht nur unter dem Gesichtspunkt betrachtet werden, frühzeitig auf die Bedürfnisse der Benutzer zu reagieren. Die Projekterfahrungen bestärken unsere Sichtweise, Prototyping als Hilfsmittel in einem Lernprozeß aller beteiligten Gruppen zu verstehen. Dies bedeutet, daß Anwender und Benutzer ein Verständnis von dem zugrundegelegten Entwicklungsmodell und von der gewählten Vorgehensweise haben müssen. Ohne

dieses Verständnis bleiben die Aufgaben und Einflußmäglichkeiten der Beteiligten unklar, und es entstehen immer wieder Mißverständnisse. Auf Entwicklerseite muß die Mäglichkeit erkannt werden, durch Prototyping einen guten und "einfachen" Zugang zum bestehenden Anwendungswissen zu erhalten. Denn ohne dieses Fachwissen kann auch mit Hilfe von Prototyping keine fachlich hochwertige Anwendung erstellt werden. Eine wichtige Voraussetzung für einen effektiv gestalteten Lernprozeß ist allerdings eine mäglichst große personelle Kontinuität innerhalb des integrierten Entwicklungsteams.

Benutzerbeteiligung beim Prototyping innerhalb einer evolutionären Vorgehensweise kann nicht undifferenziert erfolgen. Verschiedene Dimensionen sind zu berücksichtigen. Da ist, wie gesagt, zu unterscheiden, ob das Softwareprojekt für ein zugeschnittenes Anwendungssystem innerhalb einer Organisation geplant ist oder für ein marktgängiges Produkt. Im ersten Fall steht der Kreis der potentiellen Ansprechpartner für das Projektteam fest, im zweiten Fall muß überlegt werden, wer auf welche Weise als potentieller Anwender oder Benutzer angesprochen werden kann. Die Vielfalt und Anzahl der beteiligten Gruppen ist ein weiterer Gesichtspunkt. So macht beispielsweise eine kleine homogene Benutzergruppe kein eigenständiges Problem beim Prototyping, dagegen erfordern dutzende von unterschiedlichen Benutzergruppen in verschiedenen (Teil-) Organisationen eigene †berlegungen bezogen auf ihre Beteiligung an einem Prototyping-Projekt. Weitere Dimensionen der Beteiligung am Prototyping-Prozeß sind z.B. die räumliche Verteilung der einzelnen Gruppen oder die Art des vorhandenen Fachwissens. Eine weite räumliche Verteilung erfordert eine eigenständige Koordination für Projekttreffen und Prototyping-Sitzungen. Liegt das Fachwissen der Anwender und Benutzer vorrangig im ingenieurwissenschaftlichen und naturwissenschaftlichen Bereich, dann kann die Arbeit mit Prototypen oft durch eine eher formal ausgerichtete Spezifikation von Systemteilen ergänzt werden, ohne daß die gemeinsame Kommunikationsgrundlage von Benutzern und Entwicklern zerstärt wird.

Wir sind uns im klaren darüber, daß in vielen Fällen die Entwickler nicht ganz leicht festzustellen kännen, wer die tatsächlichen Anwender und Benutzer eines Systems sind. Damit ist nicht nur das Problem angesprochen, daß die Identifikation der potentiellen und tatsächlichen Benutzer bei der Produktion von Softwareprodukten für den anonymen Markt ein eigenes Problem ist. Gelegentlich finden wir auch in zugeschnittenen Entwicklungsprojekten die Situation, daß die Entwickler nicht von der richtigen Benutzergruppe ausgehen oder keinen unmittelbaren Zugang zu ihnen erhalten.

Erfahrungen mit Oberflächen-Prototyping machen deutlich, daß neben den Benutzern und anderen Gruppen der Anwendungsseite auch Spezialisten für Benutzungsoberflächen involviert werden müssen. Dies sind sowohl (Software-) Ergonomen als auch Graphiker. Auch auf diesem Gebiet existiert ein eklatantes Manko. Die Entwickler von Oberflächen sind häufig der Meinung, daß sie genau wissen, wie "gute" Oberflächen aufgebaut sein sollten. Dies stimmt aber in der Regel nicht. Das daraus resultierende Problem verschärft sich aktuell durch die veränderte Rechtslage bezogen auf die Gestaltung und den Einsatz von Anwendungssoftware in

Unternehmen. Die einschlägigen Normen für die Systemgestaltung und die Berücksichtigung der arbeitswissenschaftlichen und sofware-ergonomischen Erkenntnisse werden zunehmend verbindlich. Wir wissen, daß die Einführung von neuen Systemen, die diesen Richtlinien widersprachen, in einzelnen Unternehmen verhindert wurde. Hier ergeben sich neue Forderungen und Mäglichkeiten des Oberflächen-Prototyping, da solche Prototypen bereits im Vorfeld der eigentlichen Systemeinführung entsprechen auf Konformität geprüft werden kännen.

Beim Prototyping innerhalb einer evolutionären Strategie sollte die Chance genutzt werden, unterschiedliche Personengruppen in den Entwicklungsprozeß einzu-beziehen und interdisziplinäre Teams zu bilden. Diese Teams sollten neben den Entwicklern und den Benutzern speziell für die Konstruktion von Oberflächen-prototypen auch Designer, Graphiker und Software-Ergonomen enthalten. Hier ist vor allem das Projektmanagement gefordert: das zunächst heterogene Projektteam muß systematisch integriert werden, d.h. eine gemeinsame Projektsprache und -kultur muß gefärdert werden; geeignete Benutzer und Entwickler müssen ausgewählt werden; die beteiligten Personen müssen die notwendige Zeit zum Lernen und für die Kooperation erhalten.

Daß jenseits dieser Probleme Benutzerbeteilung heute immer noch nicht selbstverständlich ist, ist für uns ein Zeichen mangelnden Problembewußtseins. Es fehlt das Verständnis für die wesentlichen Probleme, die Dynamik von Software-Projekten und auch für eine unabdingbare gemeinsame Kommunikationsbasis zwischen den beteiligten Gruppen. Ist dieses Problembewußtsein vorhanden, so ist die "Hemmschwelle" für eine Benutzerbeteiligung deutlich niedriger. Dies ist nicht nur ein Problem der Entwicklerorganisation. Wir stellen fest, daß viele Anwender-organisationen die Meinung übernommen haben, daß die Entwickler wohl schon wissen, was die Benutzer wirklich brauchen und deshalb auch "zugeschnittene" Anwendungssysteme "in Auftrag geben", in der Hoffnung, vom eigentlichen Entwicklungsprozeß mäglichst wenig betroffen zu sein.

6.1.4 Akzeptanz

In der Praxis treffen wir immer wieder auf Vertreter des DV-Management, die Benutzerbeteiligung und Prototyping für eine "Modewelle" halten. Zu dieser Haltung haben sicherlich die überzogenen Versprechungen beigetragen, die etwa im Zusammenhang mit Expertensystemen und CASE-Tools, aber auch mit Objekt-orientierung gemacht wurden. Wir haben gelegentlich die Meinung gehärt: "Diese Welle sitzen wir auch noch aus". Hier wird übersehen, daß der Weg zu Prototyping als Teil der evolutionären Systementwicklung von mehr als zwanzig Jahren Erfahrung im Software Engineering begleitet wird.

Ein weiterer Grund für eine ablehnende Haltung gegenüber Prototyping rührt oft daher, daß Prototyping mit einem nicht geplanten Vorgehen in der Software-Entwicklung gleich gesetzt wird. Wir haben die Zusammenhänge aufgezeigt und Probleme dort identifiziert, wo Prototyping nicht durch eine entsprechende Vorgehensweise unterstützt wird. Maßgeblich ist hierbei der Einsatz von anwendungsorientierten Dokumenttypen, die wie Szenarien oder Glossare für alle

beteiligten Gruppen verständlich sind. Dadurch werden erst die Analysen eines Anwendungsbereichs und eine fachliche Modellierung des zukünftigen Systems diskutierbar und bewertbar. Dazu kommt die sorgfältige Planung von Rückkopplungs- und Bewertungsprozessen. Wir haben in den Fallstudien gesehen, daß des äfteren Zeit oder Mittel für diese zentralen Aktivitäten fehlten und dadurch erhebliche fachliche und technische Probleme entstanden.

Ein in diesem Zusammenhang gelegentlich geäußerter Vorbehalt gegen das Prototyping ist, daß sich Benutzer und Entwickler durch einen Prototyp in einem frühen Projektstadium auf eine technische Läsung "fixieren", ohne daß grundsätzliche Alternativen in Erwägung gezogen werden. Dies kann dazu führen, daß ausgehend von einem Prototyp nur noch Details angepasst werden, ohne die fachliche und software-technische Gesamtlinie infrage zu stellen oder sogar zu verändern. Dieser Vorbehalt trifft sicherlich für das "rapid" Prototyping mit Interface-Buildern und anderen mächtigen Oberflächenwerkzeugen zu. Wir haben darauf hingewiesen, daß dazu die problematische Vorgehensweise "von der Oberfläche nach unten" kommt. Als Gegenmittel dienen vor allem eine sorgfältige Analyse der vorhandenen Arbeitsaufgaben und -tätigkeiten und eine Zielfindungsdiskussion, die sich an den fachlichen Aspekten und nicht an der Gestaltung einer Softwareoberfläche orientiert. Nach unserer Erfahrung ist nicht das Prototyping selbst die Gefahr, die zu diesem technischen "Tunnelblick" führt, sondern die generelle Unterschätzung des Aufwands und der Notwendigkeit einer sorgfältigen fachlichen Analyse des Anwendungsbereichs. Für den Entwurf des zukünftigen Anwendungssystems gibt es dann ebenfalls mehr Methoden als ein rein implementationsorientiertes Prototyping. Soll-Szenarien und Zukunftswerkstätten, bei denen in Rollenspielen und Gruppenarbeit eine zukünftige Neuorganisation und Umstrukturierung eines Anwendungsbereichs insgesamt entwickelt wird, kännen hier nur als Stichwärter genannt werden.

Eine Fallstudie zeigt, daß Prototyping vom Entwicklerteam durchgeführt wurde, obwohl dies vom Management nicht explizit vorgesehen war. Dies kann folgende Gründe haben: Der Begriff Prototyping ist im Management negativ besetzt, da er als Synonym für folgende Herangehensweise steht: "Wir konstruieren etwas, ohne genau zu wissen, was fachlich notwendig ist. Notfalls werfen wir es weg". Diese Meinung finden wir bei "Prototyping-geschädigten" Managern. Grundlage sind in der Regel Erfahrungen, die aus Projekten stammen, bei denen "Prototyping" nicht methodisch (also im hier vertretenen Sinne) durchgeführt wurde, sondern als Etikett für diese Taktik des "Durchwurstelns" mißbraucht wurde. Was neben der Vorgehensweise für das Gelingen von Prototyping-Projekten von großer Bedeutung ist, sind die bereits angesprochenen geeigneten Managementstrategien. Wir haben bereits in einer früheren Studie darauf hingewiesen, daß ein solches Projekt nicht allein mit den konventionellen Mitteln der Projektführung gesteuert werden kann. Schlechte Erfahrungen in diesem Bereich kännen ebenfalls beim Management zur Ablehnung von Prototyping führen.

6.2 Prototypen als Produkt

Wir haben Prototypen als Produkte des Prototyping-Prozesses sowohl konzeptionell als auch in der Projektpraxis diskutiert. Dabei haben wir festgestellt, daß sich die verschiedenen Prototyparten gut für eine Klassifizierung der unterschiedlichen Projektaktivitäten eignen. Zudem lassen sich Verbindungen zwischen den Prototyparten und Architekturkonzepten erkennen.

6.2.1 Systemumgebung

Anwendungsentwicklung findet zunehmend in einer sich verändernden Systemumgebung statt. Wenn wir heute über interaktive Software sprechen, meinen wir vorrangig interaktive Software auf grafischen dezentralen Systemen am Arbeitsplatz ? ob dies jetzt Arbeitsplatzrechner im Sinne von PCs oder (UNIX-) Workstations sind, oder ob sie heute zu den Laptops oder Notebooks gerechnet werden. Bei dieser von uns vorrangig betrachteten Form von Anwendungssystemen wachsen die Anforderungen an fachliche Stimmigkeit und die benutzungsgerechte Handhabung.

Fachliche Stimmigkeit umfaßt hier sowohl die fachliche Logik also die "interne" Funktionalität als auch die an der Oberfläche sichtbare Begrifflichkeit und Darstellung. Dazu kommt die Handhabung als die Art und Weise, wie Benutzer mit dem System umgehen, mit ihm "hantieren" kännen. Hierbei kommen dann auch die unterschiedlichen Ein- und Ausgabemedien zum tragen und es stellt sich die Frage, wie gut ein System für die Erledigung der entsprechenden fachlichen Aufgaben "in der Hand liegt". Damit wird auch für die Softwareentwicklung die Dualität von Form und Inhalt deutlich. Dies bedeutet, daß Oberflächengestaltung heute nicht mehr ein nachrangiger "Aufsatz" auf ein "an sich" funktionstüchtiges System sein kann, sondern daß sich Handhabung, Darstellung und fachlich-logische Aspekte gegenseitig bedingen. Hier kann u.E. nur das experimentell empirische Vorgehen des Prototyping weiterführen.

Dazu kommt, daß die formalen Verfahren der Spezifikation für interaktive Anwendungssoftware besonders schlecht greifen. Es gibt u.E. absehbar keinen Formalismus, um reale interaktive Systeme mit ihren Oberflächenkomponenten formal zu spezifizieren. Das heißt, daß ein systematischer formaler Weg von der fachlichen Spezifikation zum korrekten lauffähigen Softwaresystem schon konzeptionell nicht gangbar ist. Nur in wenigen eingeschränkten Bereichen (wie der Entwicklung standardisierter Datenbankanwendungen) ist es darüberhinaus mäglich, aus dem fachlich-logischen Anwendungsmodell eine brauchbare Oberfläche zu generieren. Auch dies macht Prototyping zur unumgänglichen Projektaktivität.

6.2.2 Prototyparten und Oberflächenprototypen

Wir haben einleitend gesagt, daß sich die Einteilung in Prototyp-Arten für die Klassifikation von Projektaktivitäten bewährt hat. Die untersuchten Projekte lassen

einen Trend zu einer Differenzierung beim Bau von Prototypen erkennen: Es wird nicht mehr nur ein Prototyp gebaut, sondern die unterschiedlichen Prototyparten werden gezielt eingesetzt. Einige Projekte zeigen, daß es billiger und insbesondere fachlich besser ist, ein Ensemble verschiedener Prototypen zu bauen. Dadurch kännen die verschiedenen Prototyparten besser auf die jeweils im Projekt relevanten Fragestellungen zugeschnitten werden.

Viele der Fallstudien zeigen, daß die ersten Prototypen entweder direkt zur Projektakquisition oder zumindest als Demonstrationsprototypen für das mittlere bzw. obere Management bestimmt sind, um hier ein Verständnis für das zu entwickelnde System zu schaffen. Diese Prototypen werden allerdings oft nicht ausreichend mit den Benutzern diskutiert, wobei dies allerdings auch nicht ihr vorrangiger Zweck ist. Interessant in diesem Zusammenhang ist, daß Prototyping zu Akquisitionszwecken meist eine hähere Akzeptanz beim Management hat als sein Einsatz bei der anwendungsorientierten Entwicklung in integrierten Teams.

Während die begriffliche Differenzierung nach Prototyparten nützlich ist, hat die konzeptionelle und analytische Betrachtung der Projekte gezeigt, daß Oberflächen-Prototyping kein homogener Begriff ist. Die dabei entwickelten Prototypen kännen als reine Demonstrationsprototypen oder als funktionale Prototypen ausgelegt sein. Ebenso differenziert ist der Prozeß selbst: Bei genauer Betrachtung finden wir alle drei Arten des Prototyping ? explorativ, experimentell und evolutionär.

Was ist also Oberflächen-Prototyping letztlich? Wir haben gesehen, daß dabei das Anwendungsgebiet und die verwendete DV-Technologie eine große Rolle spielen. Im Bereich der von uns eingegrenzten interaktiven Anwendungssysteme, die wir heute sowohl bei Informationssystemen als auch in der Prozeßsteuerung finden, kommt dem Zusammenspiel von fachlicher Komponente und Oberfläche eine besonders große Bedeutung zu. Dazu kommt, daß sich dieses Zusammenspiel den bisher propagierten formalen Ansätzen der Softwaretechnik entzieht, und daß daher ein besonders großer Bedarf nach experimentellen Ansätzen und Rückkopplung mit den Benutzern besteht. In diesem gesamten Umfeld kommt solchen Prototypen eine besondere Rolle zu, die Oberfläche und Handhabung gemeinsam modellieren ? also solche Prototypen, die wir als Oberflächen-Prototypen bezeichnet haben.

Im Sinne des gerade Besprochenen ist es besonders wichtig, reine Oberflächen-Layouts (Mock Ups) von Oberflächenprototypen zu unterscheiden. Oberflächen-Layouts sind als Entwurfsskizze im Diskussionsprozeß des Entwicklerteams oft nützlich. Sie sollten aber ihren Skizzencharakter behalten und als Wegwerfprodukte (im Sinne der Code-Wiederverwendung) konzipiert sein. Oberflächen-Layouts sind immer dann problematisch, wenn sie in der Diskussion mit Benutzern eingesetzt werden, da die Aussagen, die auf ihrer Grundlage über das zukünftige System gewonnen werden kännen, wenig zuverlässig sind. Solche Oberflächen-Layouts haben wie Demonstrationsprototypen den großen Nachteil, daß bei ihnen der fachliche Anteil "im Kopf" der Benutzer oder Entwickler mitgedacht werden muß, wenn sie auf das Zusammenspiel von fachlicher Komponente und Oberfläche untersucht werden sollen. Zudem muß beachtet werden, daß mit Oberflächen-Layouts oft ein Eindruck vom Zielsystem erweckt wird, der dann nicht realisiert

werden kann. Oberflächenprototypen sollten neben der Oberfläche zumindest eine rudimentäre fachliche Komponente zeigen. Erst dann ist es mäglich, die Handhabbarkeit eines Prototyps zu evaluieren, denn "reine" Handhabung, ohne eine inhaltlich fachliche Basis, kann nicht sinnvoll bewertet werden.

6.2.3 Architektur und Designmuster

Die zunehmend wichtiger werdende Diskussion um moderne Softwarearchitekturen hat auch Auswirkungen auf das Prototyping. Da ist zunächst der starke Einfluß, den die Objektorientierung auf die Konstruktion interaktiver Software genommen hat. Wir haben darauf hingewiesen, daß praktisch alle graphischen Fenstersysteme nach objektorientierten Gestaltungsprinzipien entworfen worden sind, auch wenn sie dann letztendlich konventionell modulorientiert implementiert wurden. Dazu kommt, daß zur Bewältigung der Komplexität umfangreicher interaktiver Anwendungen kein anderes Entwicklungsparadigma geeignet erscheint. Deshalb wird heute verstärkt die Frage an Werkzeuge zum Oberflächen-Prototyping gerichtet, in welchem Umfang sie in ein solches objektorientiertes Entwicklungsumfeld passen. Bis auf die Application Frameworks oder einige wenige Interface Builder wie VisualWorks kann aber aus unserer Sicht noch keine wirklich positive Antwort gegeben werden.

Damit stellt sich für uns die Frage, welche von den in einem Projekt entwickelten Prototypen überhaupt die Architektur und die software-technischen Grundprinzipien des zukünftigen Anwendungssystems modellieren müssen. Wenn diese †bereinstimmung gewünscht wird, dann haben wir bei einigen Werkzeugen für das Oberflächen-Prototyping Probleme identifiziert. So sind z.B. Oberflächenprototypen, bei denen werkzeugbedingt die fachliche Komponente in der Form von interpretierbaren Programmfragmenten eng in die Oberflächenkomponente integriert ist, nur begrenzt für große und komplexe Anwendungen ausbaufähig.

Ein weiterer wesentlicher Aspekt zum Thema (objektorientierter) Software-Architekturen wird unter dem Stichwort "Designmuster" diskutiert. Es geht darum, bestimmte wiederkehrende Muster bei der Läsung ähnlicher Probleme so zu beschreiben, daß daraus eine "Sprache" für die Software-Entwicklung unter Verwendung "vorgefertigter" Architekturkonzepte entsteht. Als Beispiel sei der in dieser Studie wiederholt angesprochene Mechanismus von Ereignis und Reaktion in Systemen mit Ereignisbehandlung genannt. Zumindest für Prototyping-Werkzeuge, die bei der Arbeit mit Application Frameworks eingesetzt werden sollen, wird es ein relevantes Auswahlkriterium werden, ob sich mit solchen Werkzeugen Prototypen konstruieren lassen, die der gewünschten Architekur entsprechen. Diese Architektur muß konform zu den im Entwicklerteam ausgewählten Designmustern sein.

6.4 Prototyping und Werkzeugunterstützung

Für den Einsatz der verschiedenen Werkzeuge ist neben der vorrangigen konzeptionellen Klarheit sehr wichtig, ihren Einsatzschwerpunkt zu kennen, d.h. den Typ von Anwendung, für den das jeweilige Werkzeug entwickelt worden ist. Denn

entgegen anderer Behauptungen, besonders von Seiten der Hersteller, sind auch die von uns untersuchten Werkzeuge zum Oberflächen-Prototyping keine Universalwerkzeuge. Wir haben mehrfach darauf hingewiesen, daß die Arbeit mit den jeweiligen Werkzeugen sehr schnell unhandlich und aufwendig wird, wenn sie im Grenzbereich ihres Einsatzschwerpunktes eingesetzt werden. Dabei kann der Einsatzschwerpunkt sich sowohl auf den Anwendungsbereich als auch auf die verwendete Entwurfsmetapher (z.B. "elektronischer Karteikasten") beziehen.

6.4.1 Grenzen des Werkzeugeinsatzes

Es hat sich in unserer Studie wieder einmal deutlich gezeigt, daß die Hoffnung trügerisch ist, mit Hilfe von Werkzeugen grundlegende konzeptionelle Probleme der Anwendungsentwicklung zu läsen. Grundlegend sind etwa die Sichtweise und ein dazu passendes Leitbild, die Wahl einer geeigneten Methode und Entwicklungsstrategie sowie die Analyse der Anwendungssituation. Werkzeuge kännen immer nur unterstützend auf einer vorhandenen konzeptionellen und methodischen Basis eingesetzt werden. Wenn diese Basis fehlt, kann der Einsatz von Werkzeugen mehr schaden als nützen. Gefahren eines verfehlten Werkzeugeinsatzes zeigen sich vor allem an zwei Stellen:

´ Zum einen kann die rasche Erstellung von Oberflächen-Prototypen darüber hinwegtäuschen, daß die fachlich konzeptionelle Grundlage fehlt. Auch in diesem Sinne warnen wir vor einem "Rapid Prototyping". Erfahrungen zeigen, daß Prototypen auf dieser mangelnden fachlichen und oft auch kommunikativen Grundlage nicht "konvergieren", d.h. daß in den auswertenden Diskussionen mit den Benutzern immer neue und andere Anforderungen aufkommen. Dies ist ein deutliches Zeichen, daß keine gemeinsame Basis für ein Softwareprojekt vorhanden ist (wir weisen nochmals auf unsere Anmerkungen zu einer neuen Projektkultur hin).

´ Zum zweiten kännen neue Werkzeuge auch als Argument für das DV-Management gelten, sich nicht mit grundsätzlichen Problemen der Anwendungsentwicklung auseinanderzusetzen. Auch hier sehen wir eine "Mitschuld" von Werkzeuganbietern und einigen Beratungsunternehmen, die den Eindruck erwecken, durch Werkzeugeinsatz ließen sich unter Beibehaltung der traditionellen Vorgehens- und Sichtweise grundlegende Probleme meistern. Kennzeichen dieser Problemlage sind vällig illusionäre Terminplanungen unter besonderer Vernachlässigung einer ausreichenden Analyse des Anwendungsbereichs. Dazu kommt oft eine personelle Unterdeckung des Projekts. Dies alles geschieht mit dem Hinweis auf die zu erwartende ungeheure Effizienz des neuen Werkzeugs. Dabei wird gelegentlich sogar auf eine eigene Einarbeitungsphase für ein solches Werkzeug verzichtet, da dies ja für die einfache Anwendung beim Rapid Prototyping gedacht sei.

6.4.2 Auswirkungen auf den Prozeß

Viele Werkzeuge zur Obenflächenentwicklung haben Auswirkungen auf den Entwicklungsprozeß weit jenseits der Konstruktion eines Oberflächenprototyps. Besonders 4G-Systeme legen explizit oder implizit eine bestimmte Vorgehensweise

und eigene Modellierungskonzepte fest. Dies bedeutet für viele Entwicklerorganisationen eine erhebliche organisatorische und strategische Umstellung. Neben den hohen Kosten, die dies zur Folge hat, darf nicht übersehen werden, daß dadurch eine neue Form von Herstellerabhängigkeit eingegangen wird. Wir haben erlebt, daß Entwicklerorganisationen sich auf der einen Seite mühsam von der Bindung an einen Hardwarehersteller geläst haben, um andererseits sich nicht minder intensiv einem Werkzeughersteller zu verpflichten.

Bezogen auf den Einsatz von Prototyping-Werkzeugen wollen wir vor der Illusion warnen, daß die Veränderung und Weiterentwicklung eines Prototyps in der Diskussion zwischen Benutzern und Entwicklern vor Ort interaktiv am Bildschirm erfolgen kann. Dies mag für einzelne Aspekte der Oberflächengestaltung der Fall sein. Aber schon bei fachlichen oder grundsätzlich konzeptionellen €nderungen ist dies nicht mehr in "Realzeit" mäglich und im Regelfall – nach unserer Erfahrung – auch nicht erforderlich. Sehr viel wichtiger ist, daß die Entwickler verstehen, welche €nderungen erwünscht sind, um diese dann in absehbarer Zeit zu realisieren. Die eigentlichen Prototyping-Sitzungen sollten der Diskussion und dem Ausprobieren vorbehalten sein. Im übrigen sind nur wenige Prototyping-Werkzeuge so gebaut, daß sie extrem kurze €nderungszyklen überhaupt zulassen.

6.4.3 Auswahl und Einsatz

Die Auswahl des für den Anwendungsfall richtigen Werkzeugs ist differenzierter zu sehen als dies in einigen Projekten geschehen ist. Zum Teil wurde - aus fehlender Marktkenntnis - nicht das Optimum der vorhandenen Werkzeuge eingesetzt. Als Beispiel kännen die Smalltalk-Projekte gelten. Für den angestrebten Zweck hätte es eine ganze Reihe von Werkzeugen gegeben (z.B. Interface-Builder, Klassenbibliotheken etc.), die das Oberflächen-Prototyping wesentlich vereinfacht hätten. Dabei muß differenziert werden: Für standardisierte, d.h. häufige Anwendungen sind entsprechende Oberflächenkomponenten vorhanden und werden zusammen mit gut einsetzbaren Werkzeugen angeboten. Aber überall dort, wo maßgeschneiderte Läsungen komplexe oder neue Oberflächenkomponenten erfordern, ist auch heute noch "Handarbeit" gefragt. Dann erweisen sich Werkzeuge, die die Programmierung von Oberflächen erleichtern, gegenüber Werkzeugen zur Komposition parametrisierbarer Bausteinsammlungen als überlegen.

Die Auswahl der Prototyping-Werkzeuge ist keiner festen Gruppe zuzuordnen. Natürlich ist dies oft eine Aufgabe der Entwickler. Daneben gibt es aber eigene Methoden- und Werkzeuggruppen, die in einer Entwicklerorganisation für Auswahl und Installierung aller Entwicklungswerkzeuge zuständig sind. Hier ist oft ein genauer Blick angesagt. Wir kennen Fälle, in denen mit dem Argument der unternehmenseinheitlichen Softwarelandschaft von einer solchen Abteilung generell der Einsatz von Prototyping-Werkzeugen unterbunden wird, was in der Konsequenz das Prototyping unmäglich macht. Gelegentlich haben auch die Kunden als Auftraggeber eine dezidierte Vorstellung über den Werkzeugeinsatz. In der Regel ist dies eher nachteilig, da diese Vorstellung in den seltensten Fällen auf den fachlichen und software-technischen Notwendigkeiten des jeweiligen Projektes beruhen. Wir befürworten, die Werkzeugauswahl in die Kompetenz der einzelnen

Entwicklungsprojekte zu geben, die ihre Wahl allerdings mit den projektübergreifenden Randbedingungen der Entwicklungsorganisation abstimmen müssen.

Die in den Projekten verwendeten Werkzeuge wurden i.a. im intendierten Einsatzschwerpunkt eingesetzt. Dies gilt jedoch nicht für die Hypertext-Systeme, die häufig nicht als elektronischer Karteikästen, sondern als flexible Werkzeuge zur Oberflächengestaltung verwendet werden.

Der Aufwand für die Einarbeitung in das Werkzeug und in die Methodik wird häufig nicht vom Projektaufwand getrennt. Dies kann dazu führen, daß der Prototyping-Ansatz gelegentlich als sehr aufwendig eingeschätzt wird. Zwei Einflußgräßen sind hier zu nennen. Zum einen haben viele der von uns untersuchten Entwicklerorganisationen noch keine große Prototyping-Erfahrung und setzen Werkzeuge zum erstenmal ein. Der zweite Grund hängt mit dem ersten zusammen. In vielen Organisationen wird Anwendungsentwicklung noch nicht vorrangig als Lern- und Kommunikationsprozeß, sondern als technischer Entwicklungsprozeß gesehen. Deshalb werden Aufwendungen für das Erlernen neuer Methoden und die Einarbeitung in neue Werkzeuge nicht richtig eingeschätzt und tendenziell als negativ betrachtet.

6.5 Trends und Fazit

Schon bei der Vorstellung der einzelnen Werkzeugarten in diesem Bericht hat sich gezeigt, daß tendenziell die Unterschiede zwischen den einzelnen Werkzeugarten verschwinden. Interface-Builder, Application Frameworks, 4. Generationssysteme mit (partieller) interpretativer Ausführung verschmelzen. Beispiele sind der NeXT Interface-Builder oder die kommerziellen Smalltalk-Systeme (VisualWorks, PARTS, VisualAge etc.).

Diese Kombination von aufeinander abgestimmten Werkzeugen führt dank der integrierten Interface-Builder zu einfach konstruierbaren Oberflächen-Prototypen, zur flexiblen Erweiterbarkeit dank der Application Frameworks und, aufgrund der entsprechenden Abfragesprachen zur standardisierten Datenabfrage, -Anzeige und Manipulationsfunktionalität.

Die Analyse der Interface-Builder zeigt einen interessanten Aspekt: Offenbar gibt es mittlerweile ein konsoldiertes Erfahrungswissen darüber, wie die elementaren Elemente einer grafisch interaktiven Oberfläche aussehen sollten. Trotz aller Unterschiede im Detail lassen sich quer zu den unterschiedlichen Fenstersystemen und Rechnerfamilien sehr ähnliche Grundbausteine feststellen. Die Differenzierungen dieser Werkzeuge setzen bei der jeweiligen "Philosophie" bezogen auf die fachlichen Komponenten ein, d.h. welche fachlichen Anteile sind schon in den Oberflächenelementen enthalten und wie werden weitere fachliche Anteile hinzugefügt. Wir erwarten in der Zukunft, daß sich die unterschiedlichen Anwendungsbereiche expliziter auf die Zusammenstellung fachlicher Bausteine und dazu passenden komplexen Oberflächenelementen auswirken. Beispielsweise gehen einige der beschriebenen Application Frameworks in diese Richtung.

Wir erwarten eine klarere Trennung von offenen Entwicklungsumgebungen und Anwendungssoftware. Dies bedeutet zum einen, daß Application Frameworks angeboten werden, in die verschiedene miteinander kommunzierende interaktive Komponenten eingehängt werden kännen. Dann kännen Werkzeuge zum Oberflächen-Prototyping sinnvoll mit anderen Werkzeugen zur Modellierung und zur Implementierung der fachlichen Komponenten kombiniert werden. Zum anderen werden Entwicklungsumgebungen nicht mehr gleichzeitig die Laufzeitumgebung des Prototypen oder des Zielsystems darstellen. Diese Tendenz hat sich etwa schon beim Smalltalk-System gezeigt.

Werkzeuge zur Oberflächenentwicklung werden tendenziell sowohl architektonisch gut-strukturierte Programmtexte in einer Hochsprache generieren als auch wieder auf solchen Programmtexten aufsetzen kännen. Damit entfallen zwei derzeit recht gravierende Probleme: Die mangelnde Mäglichkeit, generierte Oberflächen als verständliche Bausteine für die Softwareentwicklung auch jenseits von Prototypen zu verwenden und die Schwierigkeiten, die die €nderung von Oberflächen bereitet, wenn sie um fachlichen Komponenten ergänzt worden sind.

Die sich immer stärker durchsetzende Idee des Client-Server-Computing gibt der Trennung von Oberfläche und fachlicher Anwendung eine neue Dimension. Wir sehen verschiedene Tendenzen: Zum einen werden unterschiedliche fachliche Funktionalitäten als gekapselte Dienste angeboten werden, die dann in der Anwendung in spezifische fachliche und Oberflächenkomponenten integriert werden. Die heutigen Application Frameworks werden sich in diesem Sinne verändern. Neben White Box Frameworks, die durch Reimplementierung spezialisiert werden müssen, werden in Zukunft auch Black Box Frameworks treten, die durch geeignete Werkzeuge oder Umgebungen zu spezifischen Anwendungen konfiguriert und komponiert werden kännen. Die Konstruktion eines passenden Oberflächenprototyps wird dann eine Teilaufgabe bei der Konfigurierung einer solchen Anwendung sein. Systeme dieser Art werden derzeit aber erst in Forschungseinrichtungen erprobt.

Was kännen wir nach Abschluß der Studie und mit Blick auf die aktuelle Diskussion über Stand und Zukunft des Prototyping sagen?

Zunächst ist erstaunlich, in welch großem Umfang die industrielle Software-Entwicklung die Erkenntnisse und Erfahrungen aus mehr als einem Jahrzehnt Praxis mit Prototyping ignoriert. Noch immer sind die Projekte, in denen auf einer halbwegs soliden konzeptionellen Grundlage Prototyping eingesetzt wird gegenüber den konventionell und teilweise konzeptionslos durchgeführten Projekten in der verschwindenden Minderzahl. In Anbetracht der heftigen Diskussion um neue Methoden und Werkzeuge ist das ein auf den ersten Blick unverständlicher Umstand, da die Ergebnisse von Prototyping-Projekten durchweg als ermutigend zu bezeichnen sind. Schauen wir allerdings in das organisatorische Umfeld, wird dieses Mißverhältnis verständlicher. Dann stellen wir nämlich fest, daß die westlichen industriellen Großunternehmen generell in einer Strukturkrise stecken. Wir sehen enge Verbindungen zwischen den Ansätzen des Business Process Reengineering und den Ideen der evolutionären und anwendungsorientierten Softwareentwicklung.

Daher befürchten wir, daß sich eine veränderte Denkweise in den Unternehmensleitungen insgesamt durchsetzen muß, ehe sich an der Software-Entwicklung auf breiter Front etwas entscheidendes ändert.

Ergänzend zu dieser, eher pessimistischen Einschätzung sehen wir die Situation im Forschungs- und Wissenschaftsbereich. Zumindest für den deutschsprachigen Raum gilt, daß Prototyping und damit Ansätze zur experimentellen und partizipativen Softwareentwicklung von der vorherrschenden Informatik als unwissenschaftlich und "soft" abqualifiziert werden. Nach wie vor wird dort die Hoffnung gehegt, durch umfassende formale Methoden eine vollständige Spezifikation der Systemanforderungen in korrekte Software überführen zu kännen. Auch hier sind Ansätze, die sich um eine Komplementarität von formalen Methoden und experimentell evolutionärer Vorgehensweise bemühen, eher die Ausnahme.

Erfreulicher sieht das Bild aus, wenn wir die Unternehmen betrachten, die sich auf aktuelle Trends wie Client-Service Computing, Objektorientierung und Software für kooperative, verteilte Arbeit spezialisiert haben. In diesen Bereichen sind Prototyping und die damit verbundenen Entwicklungsstrategien zur Alltagspraxis geworden. Auch DV-Abteilungen von Organisationen, die verstärkt auf kundenzentrierte Dienstleistungen umstellen, haben diesen Trend erkannt. Von diesen Bereichen werden neue Impulse für das Prototyping ausgehen. Wir erwarten, daß sich hier die wechselseitige Beziehung von Entwicklungsstrategie und Unternehmensorganisation am stärksten zeigen wird.

Prototyping für Anwendungssysteme ist heute überwiegend Oberflächen-Prototyping. Dies ist schon durch die veränderte DV-Landschaft eine Notwendigkeit geworden. Dazu kommt, daß Benutzer heute durchgängig interaktive Anwendungssoftware in dem von uns beschriebenen Rahmen fordern. Die Mittel zur Realisierung von Oberflächen-Prototyping sind vorhanden. Die Werkzeugunterstützung hat sich in den letzten Jahren wesentlich verbessert, auch wenn für einzelne Rechnerfamilien ein breites Angebot noch aussteht. Generell stellen wir fest, daß die softwaretechnischen Mäglichkeiten der Umsetzung in der Praxis noch weit voraus sind. Dieser Trend kännte sich in der nächsten Zeit noch verstärken. Integrierte Entwicklungswerkzeuge sowie die Verwendung von Designmustern und dedizierten Anwendungsbibliotheken sind in ihren Mäglichkeiten für das Prototyping bei weitem noch nicht ausgeschäpft.

Glossar

4GL - 4. Generationssprache
Abfrage- und Auswertungssystem, mit dem Benutzer und Entwickler einfach auf eine Datenbank zugreifen kännen. 4. Generationssprachen sind Bestandteile von *4. Generationssystemen*.

4GS - 4. Generationssystem
Vollständige Entwicklungsumgebung für Informationssysteme auf Basis von Datenbank-Managementsystemen. Dazu gehären u.a. Editoren, um das Datenmodell und die Benutzungsoberfläche zu erstellen, Programmiersprache(n), Report-Generatoren und das Datenbank-Managementsystem selbst.

Application Framework
Objektorientiert konstruierte Standardanwendung. Ein Framework besteht aus objektorientierten Basisklassen, die *Oberflächenkomponenten* und andere Dienstleistungen realisieren und aus Klassen, die die eigentliche fachliche Standardanwendung implementieren. Im Gegensatz zu einer *Toolbox* stellt ein Framework nicht nur Bausteine, sondern auch eine flexibel erweiterbare Standardarchitektur zur Verfügung.

Botschaften
Botschaften im objektorientierten Sinn dienen zur Kommunikation zwischen Objekten. Sie werden an Objekte geschickt und läsen dort das Ausführen einer Methode aus. Sie sind den Prozeduraufrufen in tradionellen imperativen Sprachen vergleichbar.

Callback
Werden als eine Variante der Ereignisbehandlung bei interaktiven Systemen verwendet, um verschiedene Anwendungsteile, wie z.B. die *Oberflächenkomponente* und die *fachliche Komponente*, zu integrieren. Dabei gibt die eine Komponente der anderen bekannt, welche ihrer Funktionen als Reaktion auf ein *Ereignis* aufgerufen werden soll.

Design-Muster
Abstrakte Beschreibung einer Software-Architektur. In objektorientierten Systeme legen Design-Muster semantische Beziehungen auf Klassenebene mit Hilfe der Vererbungs- und Benutzt-Beziehung fest. Verschiedene Aspekte von *Application Frameworks* kännen im Normalfall durch Design-Muster beschrieben werden.

Direktmanipulative Interaktion
Interakative Manipulation von Oberflächenelementen und den durch sie repräsentierten fachlichen Elementen mit einer Maus oder ählichen Eingabegeräten. Im Gegensatz dazu bilden Kommandosprachen oder ähnliche indirekte Mechanismen.

Entwicklungsumgebung
Werkzeuge für das Oberflächen-Prototyping kännen aus einer Entwicklungs- und einer *Laufzeitumgebung* bestehen. In der Entwicklungsumgebung wird die Benutzungsoberfläche der Anwendung erstellt. Abhängig vom eingesetzten Werkzeug wird in der Entwicklungsumgebung zudem die *fachliche Komponente* einer Anwendung beschrieben.

Entwurfsmetapher
Bildliche oder sprachliche Vorstellung, die als Anleitung bei der Gestaltung eines Systems dient; z.B. "Werkzeug" oder "intelligenter Assistent".

Ereignis (Event)
Aktion oder Aktivität, die in Softwaresystemen repräsentiert und behandelt wird. Man unterscheidet zwischen Systemereignissen und anwendungsspezifischen Ereignissen. Beispiele für Systemereignisse sind das Drücken eines Kopfes, das Selektieren eines Listenelements oder die Ankunft einer Nachricht von einer anderen Anwendung. Beispiele für anwendungsspezifische Ereignisse sind das Anlegen eines Kontos oder das Verschieben eines graphischen Objektes in einem Graphikeditor.

Ereignisbehandlung (Event Handling)
Ist die Art und Weise, wie Komponenten eines interaktiven Systems auf *Ereignisse* reagieren. Wird im allgemeinen durch *Event Handler* durchgeführt.

Event Handler
Ist eine software-technische Komponente, die *Ereignisse* analysiert, auf sie reagiert oder deren Behandlung ausläst. Event Handler werden beim Oberflächen-Prototyping häufig in einer werkzeugspezifischen *Skriptsprache* implementiert.

Fachliche Komponente
Alle Komponenten einer Anwendung, die fachliche Aspekte realisieren. Weitere Komponenten sind die Datenbank- und die *Oberflächenkomponente*.

HyperCard-System
Werkzeug, das auf dem Modell des elektronischen Karteikastens basiert. Die Elemente eines Karteikastens sind Karten, die Texte, Grafiken oder andere Oberflächenelemente enthalten kännen. Karten und ihre Elemente kännen mit Hyper-Links untereinander verbunden werden. Mit einer *Skriptsprache* kännen für jede Komponente *Event Handler* implementiert werden.

Interface-Builder
Werkzeug mit dem Benutzungsoberflächen auf hohem Abstraktionsniveau in textueller oder *direkt-manipulativer* Form konstruiert werden kännen. Interface-Builder unterstützen die Definition von *Oberflächenelementen* sowie deren Verhalten bei der Handhabung.

Laufzeitumgebung
Ermäglicht, die in der *Entwicklungsumgebung* beschriebenen Benutzungsoberflächen darzustellen und die *fachliche Komponente* auszuführen. Modifikationen an einer bestehenden Anwendung sind in der Laufzeitumgebung nicht mäglich.

Leitbild
Grundlegende Idee oder Vorstellung, die Anleitung gibt, wie Gegenstände bei der Analyse eines Anwendungsbereiches als relevant erkannt werden und wie das angestrebte System und die zukünftigen Arbeitsformen aussehen kännen.

Look & Feel
Darstellungs- und *Umgangsformen* der *Oberflächenelementee*eines Fenstersystems.

modal, multimodal
modal - Interaktionsform, bei der der Benutzer aus einer Anzahl vorgegebener Aktionen auswählen muß (z.B. OK- oder CANCEL-Knopf wählen), bevor das System weitere Interaktion ermäglicht.
In multimodalen Benutzungsoberflächen kann der Benutzer mit dem System über verschiedene Kommunikationskanäle interagieren. Als Kommunikationskanäle kännen z.B. Sprache, Gestik, Datenhandschuhe und Virtual Reality genannt werden.

Model-View-Controler Framework
Standardarchitektur für graphische Benutzungsschnittstellen. Diese definiert drei Komponenten und deren Koordination. Die Komponenten sind die fachlichen Informationen (Model), deren graphische Repräsentation (View) und die *Ereignisbehandlung* (Controler).

Oberflächenelement, Interaktionselement
Atomare Elemente, aus denen Oberflächen aufgebaut werden kännen. Beispiele sind: ein Knopf, ein Menü oder ein Eingabefeld.

Oberflächenelementart, Interaktionstyp
Beschreibt einen Typ von *Oberflächenelementen*. Ein Oberflächenelement ist immer ein Exemplar der dazugehärenden Oberflächenelementart. Beispiel: der OK-Knopf und der CANCEL-Knopf sind Exemplare der Oberflächenelementart Knopf.

Oberflächenkomponente
Alle Komponenten einer Anwendung, die deren Oberfläche realisieren. Weitere Komponenten sind die *fachliche Komponente* und die Datenbankkomponente.

Palette
In Anlehnung an die Farbpalette, aus denen ein Künstler Farben auswählt, um ein Bild zu malen, stellt ein *Interface-Builder* die angebotenen *Oberflächenelementarten* in einer Palette dem Designer einer Oberfläche zur Verfügung. Durch Selektion mit der Maus aus der Palette erhält er ein *Oberflächenelement*, das er auf der Entwurffläche plazieren kann.

Reaktives System, Interaktives System
Reagiert auf eine Benutzeraktion, die über ein Eingabegerät erfolgt, mit einer entsprechenden Reaktion auf einem Ausgabegerät. Aktionen in reaktiven Systemen kännen dabei nur vom Benutzer und nicht vom System ausgehen. Im Gegensatz dazu stehen sogenannte ablauforientierte Systeme, bei denen die Steuerung der Anwendung vom System ausgeht.

Skriptsprache
Interpretative Programmiersprache, die dazu verwendet wird, kleine Codefragment zu schreiben, die Ereignisse behandeln wie z.B. das Dücken eines Knopfs. Solche Codefragemente werden *Event Handler* genannt. Skriptsprachen sind werkzeug-spezifisch.

Toolbox
Modul-Bibliothek, die abstrakte Datentypen zur Verfügung stellt, mit denen man *Oberflächenelemente* wie Fenster oder Menüs auf relativ hoher Abstraktionsebene handhaben kann.

UIMS (User Interface Management System)
Erlaubt *direkt-manipulativ* Oberflächen zu definieren und zu erzeugen. Dazu enthält ein UIMS einen *Interface-Builder*. Ein UIMS kann aus einer *Entwicklungs*- und aus einer *Laufzeitumgebung* bestehen.

Umgangsform, Handhabungsform
Beschreibt, wie der Benutzer mit einem *Oberflächenelement* bzw. mit allen Exemplaren einer *Oberflächenelementart* umgeht. So kann beispielsweise ein Knopf gedrückt und in einer Liste selektiert sowie geblättert werden.

Literatur

[Bischofberger 92] W.R. Bischofberger, G. Pomberger: Prototyping-Oriented Software Development - Concepts and Tools, Springer-Verlag.

[Budde 92] R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven: Prototyping - an Approach to Evolutionary System Development. Springer-Verlag Berlin, Heidelberg, New York.

L.P. Deutsch: Design Reuse and Frameworks in the Smalltalk-80 System.

[Floyd 84] Chr. Floyd: A Systematik Look at Prototyping. In Budde er al. (eds) Approaches to Prototyping, Springer-Verlag, pp 105-122.

E. Gamma, R. Marty: ET—An Editor Toolkit for Bitmap-Oriented Workstations. Research Report Nr.86.01, Institut für Informatik, Universität Zürich, 1986.

E. Gamma: Objektorientierte Software-Entwicklung am Beispiel von ET++. Springer-Verlag, 1992.

[Goldberg 83] A. Goldberg, D. Robson: Smalltalk-80 The Language and its Implementation. Addison-Wesley.

[Goldberg 84] A. Goldberg: Smalltlalk-80 The Interactive Programming Environment. Addison-Wesley.

[Grycan 93] G. Grycan, H. Züllighoven: Objektorientierte Systementwicklung - Leitbild und Entwicklungsdokumente. Informatik Spektrum, Vol. 15, No. 5, pp 264-272.

R.E. Johnson, B. Foote: Designing Reusable Classes, JOOP, June/July .

R.E. Johnson, V.F. Russo: Reusing Object-Oriented Designs. University of Illinois tech report UIUCDCS 91-1696.

[Kieback 91] A. Kieback, H. Lichter, M. Schneider-Hufschmidt, H. Züllighoven: Prototyping in industriellen Software-Projekten - Erfahrungen und Analysen, GMD-Studie Nr. 184, St. Augustin.

[Kieback 92] A. Kieback, H. Lichter, M. Schneider-Hufschmidt, H. Züllighoven: Prototyping in industriellen Software-Projekten - Erfahrungen und Analysen, Informatik-Spektrum, Vol 15, Nr. 2, pp 65 - 77.

[Lichter 93] H. Lichter, M. Schneider-Hufschmidt, H. Züllighoven: Prototyping in Industrial Software Projects - Bridging the Gap between Theory and Practice, Proceedings of the 15th ICSE, IEEE Computer Society Press, pp 221-229.

[Linton 89] M. A. Linton, J. M. Vlissides, P. R. Calder: Composing User Interfaces with Interviews. IEEE Computer, February.

[Lübbecke 93] H. Lübbecke: Prototyping als Entwicklungsmethode für ein Anwendungs(teil-)projekt, in Requirements Engineering '93: Prototyping, German Chapter of the ACM Berichte 41, B.G. Teubner Stuttgart, pp 41-48.

[Maaß 92] S. Maaß; Oberquelle: H: Perspectives and meatphors for human computer interaction. In: Floyd, C, Züllighoven, H., Budde, R., Keil-Slawik, R., (eds.): Software Development and Reality Construction. Berlin: Springer.

[Meyer 88] B. Meyer: Object-Oriented Software Construction. Prentice-Hall, New York.

[Myers 92a] B.A. Myers, M.B. Rosson: Servey on User Interface Programming. Proceedings of the SIGCHI«92, Monterey, May 3-7.

[Myers 92b] B.A. Myers: State of the Art in User Interface Software Tools. In H.R. Hartson, D. Hix (eds) Advances in Human-Computer Interaction, Volume 4, Ablex Publishing, Norwood NJ.

K.J. Schmucker: Object-Oriented Programming for the Macintosh, Hayden Book Company.

[Vlissides 89]J.M. Vlissides, M.A. Linton: Unidraw: A Framework for Building Domain-Specific Graphical Editors. Proc. of the ACM SIGGRAPH/SIGCHI User Interface Software and Technologies Conference.

A. Weinand, E. Gamma, R. Marty: Design and Implementation of ET++, a Seamless Object-Oriented Application Framework. Structured Programming, Vol. 10, No. 2.

A. Weinand: Objektorientierte Architektur für graphische Benutzungsoberflächen—Realisierung der portablen Fenstersystemschnittstelle von ET++. Springer-Verlag